AVID: Automatic Visualization Interface Designer

Mei C. Chuah

June 25, 2000
CMU-CS-00-128

School of Computer Science
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy

DISTRIBUTION STATEMENT A
Approved for Public Release
Distribution Unlimited

Thesis Committee:

Steve Roth, Co-chair
Jim Morris, Co-chair
Scott Fahlman
Jock Mackinlay, Xerox PARC
Dan Olsen

Copyright © 2000, Mei C. Chuah

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA) under Contract No DAA-1593K0005. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of DARPA or the U.S. government.

20010316 011



School of Computer Science

DOCTORAL THESIS
in the field of
COMPUTER SCIENCE

AVID: Automatic Visualization Interface Designer

MEI CHUAH

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

ACCEPTED:

THESIS COMMITTEE CO-CHAIR	**DATE** 5/3/00
THESIS COMMITTEE CO-CHAIR	**DATE** 5/4/00
DEPARTMENT HEAD	**DATE** 5/14/00

APPROVED:

DEAN	**DATE** 5/4/00

To

Claude Fennema

who inspired me and made all this possible

© Copyright 2000
MEI C. CHUAH

AVID: AUTOMATIC VISUALIZATION INTERFACE DESIGNER

MEI C. CHUAH, PH.D.
CARNEGIE MELLON UNIVERSITY 2000

Today's widespread, cheap and fast communication makes a great variety and quantity of data available to consumers. *Information presentation* addresses the important problem of packaging and visualizing this data for users in a way that facilitates understanding and analysis. Information presentations can be created by human designers or they can be automatically generated by expert computer systems. Automatic generation offers great flexibility in performing data and information analysis tasks, because new designs are generated on a case by case basis to suit current and changing future needs. This is crucial in areas or domains where it is difficult to capture beforehand all combinations of data and analysis goals desired by users, since pre-conceived human designs are then less feasible. The focus of this thesis is to improve designs generated by automatic systems and to expand the range of tasks that can be addressed by such systems. Previous work in this area dealt primarily with how data can be mapped to graphics effectively, based on established design knowledge and perceptual rules. In this thesis I expand automatic presentation design to include not only effective mapping rules but also rules describing how data may be pre-processed before it is presented. I will show that expanding automatic design in this way allows us to consider a much wider range of designs, improves the quality of automatically generated designs, and enables automatic systems to deal with larger data sets and a wider range of tasks. The addition of data pre-processing functions also allows us to include input devices in graphical presentations, thus making them more active, engaging and flexible for users. Previous work did not consider input devices because their use is limited when we consider only mapping functions in our designs. This thesis develops a framework and design strategies for expanding the quality and breadth of automatically generated information presentations. This will in turn improve the effectiveness with which computer systems can communicate data to users, facilitating understanding and analysis of a large variety of data, over a wide range of information goals.

ACKNOWLEDGEMENTS

This work was supported by DARPA. I thank my advisor Steve Roth for his friendship and steadfast support throughout my entire graduate career. I also thank my committee members Scott Fahlman, Jock Mackinlay, Jim Morris, and Dan Olsen, for their invaluable advice and help on my thesis research and document. Last but not least I thank Richard Burton, Stu Card, Steve Eick, Claude Fennema, Sebastian Grassia, Linda Horiuchi, Bonnie John, Stephan Kerpedjiev, John Kolojejchick, Joe Mattis, John Miller, Stephen North, Poe, George Robertson and everyone in my family for their help, support, and encouragement.

TABLE OF CONTENTS

CHAPTER I: INTRODUCTION I-1

I-1 METHODS: VISUALIZATION TECHNIQUES FRAMEWORK I-6
I-2 PRINCIPLES: DESIGN DIMENSIONS AND STRATEGIES FOR MEASURING THE GOODNESS OF VISUALIZATION DESIGNS I-9
I-3 SYSTEMS: AVID – AUTOMATIC VISUALIZATION INTERFACE DESIGNER I-11
I-4 PREVIOUS WORK I-13
I-4.1 Visualization Techniques Framework I-13
I-4.2 Automatic Visualization Design I-13
I-4.2.1 Automatic Visualization Systems I-14
I-4.2.2 Psychophysical Studies I-14
I-4.2.3 User Studies on Visualization Interfaces I-15
I-5 SUMMARY I-15
I-6 WALKTHROUGH I-16

CHAPTER II: VISUALIZATION TECHNIQUES FRAMEWORK A FUNCTIONAL DESCRIPTION II-17

II-1 VISUALIZATIONS & VISUALIZATION TECHNIQUES II-19
II-1.1 Object Definition Component II-22
II-1.2 Transformation Component II-24
II-2 COMPOSITION II-27
II-2.1 Object Definition Composition II-28
II-2.2 Transformation Composition II-31
II-2.3 Produce-Consumer Composition II-32
II-2.4 Partition Composition II-33
II-2.5 Summary II-34
II-3 VISUALIZATION TECHNIQUES DESIGN SPACE II-35
II-3.1 Data Transforms II-37
II-3.2 Mapping Transforms II-41
II-3.3 Graphical Transforms II-43
II-3.4 Rendering Transforms II-46
II-3.5 Summary II-48
II-4 CONCLUSION II-48

CHAPTER III: VISUALIZATION TECHNIQUES FRAMEWORK A CONCRETE INSTANTIABLE DESCRIPTION III-50

III-1 REPRESENTATION LANGUAGE III-54
III-1.1 Visualization Elements or Properties III-55
III-1.1.1 Data Concepts III-55
III-1.1.2 Graphical Objects III-56
III-1.2 Visualization Functions III-59
III-1.2.1 Object Definition Functions III-59
III-1.2.2 Transformation Functions III-60
III-1.2.2.1 Mapping Transforms III-60
III-1.2.2.2 Data and Graphical Transforms III-64
III-1.2.2.3 Rendering Transforms III-69
III-1.3 Input & Output Translation Functions III-69
III-1.4 Input-devices III-72

III-1.5 Summary *III-73*
III-2 VISUALIZATION TECHNIQUES INSTANTIATION SPACE III-73
III-3 EVALUATION OF FRAMEWORK III-78
III-3.1 Completeness *III-78*
III-3.2 Coverage *III-79*
III-3.3 Practicality *III-79*
III-3.3.1 Reduces Cost of Task Tailoring III-79
III-3.3.2 Provides a New Design Methodology III-80
III-3.3.3 Allows Systematic Exploration of the Visualization Techniques Design Space III-80
III-4 CONCLUSION III-80

CHAPTER IV: DESIGN HEURISTICS
DATA COMPUTATION VS. PERCEPTUAL MAPPING IV-82

IV-1 AN AIRLINE-SCHEDULING EXAMPLE ILLUSTRATING THE USE OF DATA TRANSFORMS IV-82
IV-2 VISUALIZATION DESIGN DIMENSIONS IV-89
IV-2.1 Articulatory Distance *IV-91*
IV-2.2 Expressive Distance *IV-93*
IV-2.3 Observational Distance *IV-96*
IV-3 DATA TECHNIQUES VS. MAPPING TECHNIQUES DESIGN GUIDELINES IV-98
IV-3.1 Accuracy *IV-99*
IV-3.2 Intermediate Tasks *IV-102*
IV-3.3 Availability of Perceptual Operations *IV-104*
IV-3.4 All to All Operations *IV-106*
IV-3.5 Task Variation on Attribute *IV-109*
IV-3.6 Task Specificity *IV-111*
IV-3.7 Summary *IV-114*
IV-4 CONCLUSION IV-115

CHAPTER V: IMPLEMENTATION
AUTOMATIC VISUALIZATION INTERFACE DESIGNER V-116

V-1 TASK INTERPRETER COMPONENT V-118
V-2 AUTOMATIC DESIGN COMPONENT V-122
V-2.1 Search Strategy *V-122*
V-2.1.1 Task Processing V-124
V-2.1.2 Data Attribute Mapping V-130
V-2.1.3 Post Design Processing V-132
V-2.1.4 Summary V-132
V-2.2 Design Constraints *V-134*
V-2.2.1 Constraint Dimension 1: Softness V-134
V-2.2.2 Constraint Dimension 2: Scope V-137
V-2.2.3 Constraint Dimension 3: Constraint Condition V-137
V-2.2.4 Summary V-142
V-2.3 Design Costs *V-145*
V-2.3.1 Articulatory Cost Structure V-146
V-2.3.2 Observational Cost Structure V-149
V-2.3.2.1 Task processing observational cost V-149
V-2.3.2.2 Data attribute mapping observational cost V-151
V-2.4 Summary *V-153*
V-3 VISUALIZATION REALIZER COMPONENT V-153
V-4 CONCLUSION V-154

CHAPTER VI: CONCLUSION & FUTURE WORK VI-156

VI-1 SUMMARY & RELEVANCY OF WORK VI-156
VI-1.1 Methods: Framework of the Visualization Creation Process VI-156
VI-1.2 Principles: Metrics & Heuristics for Measuring the Goodness of Visualization Designs VI-157
VI-1.3 Systems: AVID – Automatic Visualization Interface Designer VI-158
VI-2 SCOPE OF WORK VI-159
VI-2.1 Limitations of the Framework VI-159
VI-2.2 Limitations of the Metrics & Heuristics VI-160
VI-2.3 Limitations of the System Implementation VI-162

APPENDIX A: APPENDIX TO FUNCTIONAL VISUALIZATION TECHNIQUES FRAMEWORK (CHAPTER II) A-164

A-1 ODT DIAGRAMMATIC NOTATION A-164
A-2 EXPLORING THE SPACE OF VISUALIZATION TECHNIQUES A-164
A-2.1 Highlight Object Selection & Dynamic Query A-165
A-2.2 HomeFinder System & SDM Distance Operator A-166
A-3 CONTROL FUNCTIONS A-173

APPENDIX B: APPENDIX TO INSTANTIABLE VISUALIZATION TECHNIQUES FRAMEWORK (CHAPTER III) B-178

B-1 COMPARISON WITH PREVIOUS FRAMEWORKS B-178
B-2 DATA FLOW DIAGRAMS B-180
B-3 EXAMPLE: GENERATING AN INSTANTIATION SPECIFICATION FOR DYNAMIC QUERY SLIDERS B-181
B-4 SYSTEMATIC EXPLORATION OF THE INSTANTIATION LEVEL OF THE DYNAMIC QUERY SLIDER TECHNIQUE B-185
B-4.1 Changing the Specific Functions Used or Adding More Functions of the Same Type B-185
B-4.2 Changing the Translation Functions between Object Definition and Transformation Functions B-186
B-4.3 Changing How Control Arguments are Provided as well as the Default Designer Values B-188
B-4.4 Changing the Type of Input Devices Used within the Design B-189
B-4.5 Changing How Input Arguments are Provided to Input Devices B-190
B-5 EXPLORING THE SPACE OF VISUALIZATION TECHNIQUES B-190
B-5.1 Aggregation B-190
B-5.2 Data Drag & Drop B-191
B-5.3 Table Lens Semantic Zoom B-193
B-5.4 Summary B-194
B-6 OTHER VISUALIZATION TECHNIQUE ISSUES B-194
B-6.1 Repeating Visualization Techniques B-195
B-6.2 Integrating Visualization Techniques within a Common Workspace B-196

APPENDIX C: APPENDIX TO DESIGN HEURISTICS (CHAPTER IV) C-198

C-1 GOMS EVALUATION FOR AIRLINE-SCHEDULING TASK IN CHAPTER IV-1 ..... C-198
C-1.1 GOMS Evaluation for the Cognitive Solution of the Airline Scheduling Task C-198
C-1.2 GOMS Evaluation for the Pure Perceptual Solution of the Airline Scheduling Task C-201
C-1.3 GOMS Evaluation for the Perceptual + Data Computation Solution of the Airline Scheduling Task C-204
C-2 AIRLINE-SCHEDULING TASK DESIGN ALTERNATIVES C-204

C-3 TASK MODEL C-208
C-3.1 Logical Tasks (Logical Operators) *C-209*
C-3.1.1 Lookup C-209
C-3.1.2 Compute C-210
C-3.1.3 Find C-210
C-3.1.4 Comparison C-212
C-3.2 Task Extensions *C-214*
C-3.2.1 Task Embeddings C-214
C-3.2.2 Task iteration C-215
C-3.2.3 Task precision C-216
C-4 EXPLORING THE SPACE OF DATA TECHNIQUES AND MAPPING TECHNIQUES C-217
C-4.1 Task Structure Variation *C-218*
C-4.1.1 University Example C-218
C-4.1.2 Task Operator Variation C-220
C-4.1.3 Task Expansion C-222
C-4.1.4 Embedding Structure Variation C-225
C-4.2 Task Argument Variation *C-227*
C-4.2.1 Use of constants vs. data attribute value sets C-227
C-4.2.2 Use of known vs. unknown arguments C-229
C-4.2.3 Change in attribute or value type C-230
C-4.3 Data Set Variation *C-234*
C-4.4 Summary *C-236*
C-5 PURCHASING A CAR C-237

APPENDIX D: APPENDIX TO IMPLEMENTATION (CHAPTER V) D-246

D-1 STRUCTURAL & CONTENT MATCHING D-246
D-2 TRANSLATING A FUNCTIONAL DESIGN FROM AVID'S DESIGN COMPONENT INTO A COMPLETE SPECIFICATION D-247
D-3 VISUALIZATION REALIZER COMPONENT D-251
D-3.1 Graphical Object Realizer *D-251*
D-3.1.1 Single axis layout D-254
D-3.1.2 Non-unique positionals D-255
D-3.2 Functional Realizer *D-256*
D-3.2.1 Primitive visualization functions D-256
D-3.2.2 Connectors D-257
D-3.2.3 Input devices D-257
D-3.2.4 Composite visualization functions D-258
D-3.3 Summary & Scope *D-259*
D-4 INTERACTIVE FUNCTIONS EDITOR D-259

APPENDIX E: USING GOMS TO EVALUATE OUR AUTOMATIC DESIGN SYSTEM E-262

E-1 TASK 1: FIND TASK FINDING A "GOOD" UNIVERSITY BASED ON GRADUATION RATES AND TEST SCORES E-265
E-1.1 Design 1 ..... *E-267*
E-1.2 Design 2 ..... *E-269*
E-1.3 Design 3 ..... *E-270*
E-1.4 Design 4 ..... *E-273*
E-1.5 Design 5 ..... *E-276*
E-1.6 Design 6 ..... *E-279*
E-1.7 Design 7 ..... *E-282*
E-1.8 Summary *E-285*
E-2 TASK 2: COMPUTE TASK COMPUTE TOTAL NON-SALARY BENEFITS DISTRIBUTED BY A SET OF UNIVERSITIES E-287

E-2.1 Design 1 ..... *E-289*
E-2.2 Design 2 ..... *E-290*
E-2.3 Design 3 ..... *E-291*
E-2.4 Design 4 ..... *E-293*
E-2.5 Design 5 ..... *E-299*
E-2.6 Design 6 ..... *E-303*
E-2.7 Design 7 ..... *E-306*
E-2.8 Summary *E-310*
E-3 TASK 3: COMPARISON TASK + SIMPLE COMPUTATION EVALUATING THE RELATIONSHIP BETWEEN STATE SIZE AND VOTING RESULTS E-315
E-3.1 Design 1 ..... *E-316*
E-3.2 Design 2 ..... *E-318*
E-3.3 Design 3 ..... *E-320*
E-3.4 Design 4 ..... *E-322*
E-3.5 Design 5 ..... *E-323*
E-3.6 Design 6 ..... *E-325*
E-3.7 Design 7 ..... *E-329*
E-3.8 Summary *E-334*
E-4 TASK REFINEMENT AND SORTING E-336
E-5 WHY GOMS? E-340
E-6 CONCLUSION E-341

APPENDIX F: ENHANCING READABILITY WITH GRAPHICAL & RENDERING TRANSFORMS F-343

F-1 READABILITY PROBLEMS F-343
F-1.1 Occlusion *F-343*
F-1.2 Density *F-344*
F-1.3 Dwarfing *F-345*
F-1.4 Spatial separation *F-345*
F-2 READABILITY SOLUTIONS F-345
F-2.1 Constant Graphical Methods *F-347*
F-2.2 Additive Graphical Methods *F-347*
F-2.3 Multiplicative Graphical Methods *F-348*
F-2.4 Linear Positional Rendering Methods (Point of View Navigation) *F-348*
F-2.5 Non-linear Positional Rendering Methods (Distortion) *F-348*
F-3 APPLYING GRAPHICAL AND RENDERING METHODS TO READABILITY PROBLEMS F-349
F-3.1 Occlusion *F-350*
F-3.2 Density *F-350*
F-3.3 Dwarfing *F-351*
F-3.4 Spatial separation *F-351*
F-4 GRAPHICAL AND RENDERING TRANSFORM GUIDELINES F-351
F-4.1 Relevance of Readability Problems with respect to Tasks *F-352*
F-4.2 Continuity *F-353*
F-4.3 Individual vs. Group Readability *F-354*
F-4.4 Object spatial proximity *F-354*
F-4.5 Reversibility *F-354*
F-4.6 Learning *F-355*
F-5 CONCLUSION F-356

REFERENCES 358

LIST OF FIGURES

CHAPTER I: INTRODUCTION I-1

Figure I-1: Cognitive design for the airline-scheduling task (Note that the flights are not all shown here because the table is very large) I-2
Figure I-2: Perceptual design for the airline-scheduling task Each line represents a flight with origin and destination city mapped onto the *y-axis* and arrival and departure time mapped onto the *x-axis*. This is the best design that gets generated when ONLY mapping operations are considered by the automatic system. I.e. this is the best possible design from current state of the art systems. I-3
Figure I-3: Design generated when data processing operations are integrated into the automatic visualization system. The full data set is considered here but data transforms are applied by the automatic system to filter the data set so that only relevant flights are shown. The total downtime before the meeting for the flights from LAX to ORD is shown on the left chart and the total downtime after the meeting for the flights from ORD to BOS is shown on the right chart. I-4
Figure I-4: GOMS estimated total time for solving the airline-scheduling task for a data set of 135 flights. Detailed GOMS sequences for each design are presented in appendix C-1. ..... I-4
Figure I-5: The four phases of the visualization creation process I-6
Figure I-6: Highlight technique specification. The input-device bounding-box is used to select a set of objects. These objects (*selected-objects*) are extracted from the bounding-box device using the *get-values* function. We then get the *color* values from all of the selected objects using a subsequent *get-values* function. Finally we change all of the *color* graphical values to *red* using the *assign* function. I-7
Figure I-7: Three components within AVID I-11

CHAPTER II: VISUALIZATION TECHNIQUES FRAMEWORK A FUNCTIONAL DESCRIPTION II-17

Figure II-1: Visualization generation process consisting of four transformation classes (data, mapping, graphical, and rendering) across three different realms (data, graphical, output media) II-20
Figure II-2: Example visualization with house data. Each mark represents a *house* data concept. The *x-axis* shows *date_sold*; the *y-axis* shows *selling_price*; and *color* shows *neighborhood*. II-20
Figure II-3: A visualization technique is defined in this work to contain two components (object definition and transformation). Object definition can be achieved through enumeration or functional description. Transformation can be achieved through data, mapping, graphical, or rendering functions. The transformation functions can be further divided based on their goals. II-21
Figure II-4: Dynamic Query Sliders applied to house data. Each *bar* encodes a *house data concept*; *x-axis* encodes *date_on_market* and *date_sold*; *y-axis* encodes *house_address*. There are two dynamic query sliders [Ahlberg, 1992], one allows users to place constraints on the *num_rooms* data attribute and the other allows users to place constraints on the *selling_price* data attribute. Houses that do not fulfill constraints become non-visible as in (b). II-22
Figure II-5: Map showing the different parking lots at CMU (borrowed from http://www.cmu.edu). Clicking the *next-lot* button will cause a predefined parking lot to get highlighted red (e.g. *morewood parking*). Subsequent presses to the *next-lot* button will cause subsequent parking lots to get highlighted II-23
Figure II-6: Visualization system that allows re-mapping of data attributes to the two positional axes. Each *mark* in the visualization represents a *house* data concept. Currently the *x-axis* is set to encode *date_sold* and the *y-axis* is set of encode *selling_price*. II-24
Figure II-7: TableLens System [*Rao, 1994*] (borrowed from www.inxight.com) II-25
Figure II-8: Example rendering visualization techniques II-26

Figure II-9: Simple aggregation technique. *Text* encodes *house_address*. By using this technique users get to select a set of *house* data concepts using a *bounding-box* and aggregate or group them together to form an aggregate object (e.g. *aggregate_obj_0*). Unlike techniques in the previous section, this aggregation method utilizes multiple transformation methods including a graphical transform to highlight the selected objects and a data transform to summarize the underlying data concepts of the selection. II-27
Figure II-10: Object definition composition for the multiple constraint dynamic query technique shown in Figure II-4. The diagrammatic conventions and notations used in the specifications in this chapter and the next are described in appendix A-1. ..... II-28
Figure II-11: HomeFinder system [*Tweedie, 1994*] (borrowed from http://infoeng.ee.ic.ac.uk/~lisat/LisaDir/att.html). There are five single-axis aligned charts and a *mark* in each chart represents a *house* data concept. This system uses object definition composition to integrate the object sets selected by the three sliders and the single set of radio buttons. We then *count* the number of times a house appears in the combined set, and use this *count_attribute* to set the color for a given *house* concept. II-29
Figure II-12: Functional specification for the HomeFinder system shown in Figure II-11. ..... II-30
Figure II-13: Object definition composition on the object selection and dynamic query slider technique. The resulting technique ONLY colors those objects that are selected within the *bounding-box* as well as passes the constraint set on the *slider*. II-31
Figure II-14: Transformation composition for two different selection techniques with different visual feedback effects. The resulting technique colors and enlarges the objects selected by the *bounding-box*. II-31
Figure II-15: Transformation composition for the simple aggregation technique shown in Figure II-9. Transformation composition is used here to combine a data transform for creating the aggregate object as well as the graphical transform that highlights the objects within the aggregate, *red*. II-32
Figure II-16: P-C composition. The computed value from the producer technique (i.e. *mean selling_price*) is piped into the object definition component of the consumer technique and is used to select other objects in the visualization based on the computed *mean selling_price*. II-32
Figure II-17: P-C composition applied to the modified value-painting technique. *Text* here encodes *house_owners*. II-33
Figure II-18: Partition composition applied to the dynamic query slider technique. Here we use partition composition so that we can enlarge the *focus* objects (i.e. the objects that pass the *slider* constraint) and simultaneously contract objects in the *context-set* (i.e. objects that did not pass the *slider* constraint). II-33
Figure II-19: Two different technique descriptions that achieve the same effect. Both techniques highlight objects selected by the *bounding-box* or the *slider*, *red*. II-35
Figure II-20: The two components that form a primitive visualization technique – object definition and transformation II-36
Figure II-21: Range dynamic query technique. In the example visualizations above, each *mark* represents a *house* data concept. The *x* and *y positions* of the *marks* corresponds to the geographic location of their respective *houses*. Objects are selected here by setting constraints using the two sliders at the bottom of the interface, which allow users to set threshold constraints on the *selling_price* data attribute and the *num_rooms* data attribute. The red *bounding-box* is then drawn so that it encapsulates all of the selected objects (i.e. all objects that pass the *slider* constraints). II-40

CHAPTER III: VISUALIZATION TECHNIQUES FRAMEWORK A CONCRETE INSTANTIABLE DESCRIPTION III-50

Figure III-1: Diagrammatic representation of the five-step instantiation augmentation process for the object highlight technique. Additions made in each step are shown in gray III-52

Figure III-2: Example visualization containing house data. Each mark represents a house data concept. The *x-axis* shows the *date-sold* data attribute; the *y-axis* shows the *selling-price* data attribute, and *hue* shows the *neighborhood* data attribute III-56
Figure III-3: Example visualization of house data. Hierarchical breakdown of graphical objects in this visualization is shown in Figure III-4. *X-axis* in left-most chart shows *selling-price*; *x-axis* in middle chart shows *date-sold*, *shape* shows *neighborhood*, and *saturation* shows *salary*; *text* in right-most table shows *house-owner*. *Y-axes* for all three regions show *house-address* data attribute. III-57
Figure III-4: Container hierarchy for visualization in Figure III-3 ..... III-58
Figure III-5: Example region layout schemes (borrowed from [*Chuah, 1995*]) III-58
Figure III-6: *House* data-type to *bar* graphical-class mapping applied to the entire visualization. Both chart regions within the visualization inherit this mapping relationship. The *x-axis* of left chart shows *selling-price*; *x-axis* of right chart shows *house-lot-size* and *y-axis* of both charts show *house-address*. III-61
Figure III-7: The same visualization design as Figure III-6 except that a *house* data-type to *bar* graphical-class mapping is applied to the left region and a *house* data type to *mark* graphical-class mapping is applied to the right region III-62
Figure III-8: The same visualization design as Figure III-6 except that a *house* data-type to *mark* graphical-class mapping transform is applied to particular graphical objects in the left chart including *Woodwell-6663, Ivy-704, Penham-6828,* and *Kipling-5454* III-63
Figure III-9: SDM lift objects technique III-65
Figure III-10: SDM thin objects technique III-65
Figure III-11: Example of a dynamic query slider technique that allows users to select various *distributor* data concepts based on the number of employees (*num_people*) working at each site III-75
Figure III-12: Example dynamic query slider technique with selectable data attribute constraint III-76
Figure III-13: Example dynamic query slider technique with color and size feedback on the selected objects III-77

CHAPTER IV: DESIGN HEURISTICS
DATA COMPUTATION VS. PERCEPTUAL MAPPING IV-82

Figure IV-1: Solving the airline-scheduling task fully perceptually (Casner's solution). Each line represents a flight with origin and destination city mapped onto the *y-axis* and arrival and departure time mapped onto the *x-axis*. This is the best design that gets generated when ONLY mapping operations are considered by the automatic system. I.e. this is the best possible design from current state of the art systems IV-83
Figure IV-2: Casner's analysis of the perceptual procedure a user must perform with a visualization to achieve the airline-scheduling task (Task IV-1) IV-84
Figure IV-3: Our hybrid data transform and mapping transform design for solving airline-scheduling task. Here only the flights that fulfill the *city* and *meeting time* constraints are shown. Computation of the total downtime for the best flights is left to the user. *Time_before_meeting* is mapped to the *x-axis* of the left chart and *time_after_meeting* is mapped to the *y-axis* of the right chart. To perceptually compute the total downtime users add the shortest bar length in the left chart with the shortest bar length in the right chart. IV-85
Figure IV-4: GOMS estimated total time for solving the airline-scheduling task using a pure cognitive, pure mapping, and a hybrid data + mapping design IV-86
Figure IV-5: Visualization for finding flights with low total-downtime, low total-cost, and low duration. Because there are trade-offs that must be made among the three attributes, this task is best performed through perceptual perusal. IV-88
Figure IV-6: Interaction Framework model presented by Abowd and Beale [*Abowd, 1991*]. This framework is used to measure the effectiveness of various visualization interfaces in this work. IV-89

Figure IV-7: Breakdown of articulatory distance. Gray highlighted rectangles indicate the dimensions that are taken into account in our prototype automatic presentation system described in chapter V. IV-91
Figure IV-8: Input devices with different effectiveness properties IV-92
Figure IV-9: Breakdown of expressive distance. Gray highlighted rectangles indicate the dimensions that are taken into account in our prototype automatic presentation system described in chapter V. IV-93
Figure IV-10: Encoding house neighborhood with saturation, This encoding has low data correctness because *saturation* is an ordered graphical property while *neighborhood* is not an ordered data attribute. By using *saturation* to encode *neighborhood* we are falsely implying an ordered set of *neighborhood* values when actually there is none. IV-95
Figure IV-11: Breakdown of observational distance. Gray highlighted rectangles indicate the dimensions that are taken into account in our prototype automatic presentation system described in chapter V. IV-96
Figure IV-12: Graphic for determining the total benefits for associate professors by getting the difference between total compensation (blue bar) and total salary (red bar) IV-99
Figure IV-13: Data computation design for accurately computing total benefits for associate professors IV-100
Figure IV-14: Pure mapping design for accurately computing total benefits for associate professors IV-100
Figure IV-15: Data computation design for computing total benefits for associate professors and full professors. In this case both total benefits have been pre-computed and are shown as stacked bars. IV-103
Figure IV-16: Pure mapping design for computing total benefits for associate professors and full professors. In this case, we need to perform the entire task perceptually. Initially we must get the bar differences of the first two bars (red and green) and the last two bars (blue and purple). We must then sum up these differences to get the total benefits. IV-103
Figure IV-17: Data computation design for computing average number of teaching staff per university. In this case the average number of teaching staff has been pre-computed and the results are shown on the *x-axis*. IV-105
Figure IV-18: Pure mapping design for computing average number of teaching staff per university. In this case the average number of teaching staff must be perceptually estimated by finding an average line across each cluster of bars. IV-105
Figure IV-19: Data computation design for computing total downtime and total cost for all pairs of flights that fulfill our airline-scheduling criteria. Total downtime is pre-computed and encoded on the *x-axis* while total cost is pre-computed and encoded as *saturation*. IV-107
Figure IV-20: Mapping design for computing total downtime and total cost for all pairs of flights that fulfill our airline-scheduling criteria. *Time_after_meeting* is mapped on the *x-axis* of the left chart, *time_before_meeting* is mapped on the *x-axis* of the right chart, and *flight_price* is mapped to *saturation* in both charts IV-108
Figure IV-21: Data computation design for computing total number of votes in each state and ranking the three political parties based on the number of votes received. *Total_number_of_votes* has been pre-computed and is shown on the *x-axis* of the left chart. *Party_ranking* has also been pre-computed and is shown in the right table. IV-110
Figure IV-22: Mapping design for computing total number of votes in each state and ranking the three political parties based on the number of votes received. The *#_votes_for_Republican_party* is mapped to the *x-length* of the *red* bar, the *#_votes_for_Democratic_party* is mapped to the *x-length* of the *green* bar, and the *#_votes_for_Independant_party* is mapped to the *x-length* of the *purple* bar. Total votes can be derived by looking at the combined length of the stacked bar and party ranking can be derived by comparing the three differently colored bar lengths for each state. IV-110
Figure IV-23: Data computation design for finding the best university based on *out-of-state-tuition, graduation-rate,* and *student-faculty-ratio.* Thresholds for each condition can be entered through the three sliders and those universities that fulfill the threshold conditions are pre-computed and shown. IV-112

Figure IV-24: Mapping design for finding the best university based on *out-of-state-tuition, graduation-rate,* and *student-faculty-ratio. Student-faculty-ratio* is mapped to the *x-axis, graduation_rate* is mapped to the *y-axis,* and *out_of_state_tuition* is mapped to *saturation.* The best universities are those in the upper-left corner of the display, with low *saturation.* IV-112
Figure IV-25: Identical design as Figure IV-24 but applied to a larger data set. As a result there is significantly more occlusion making it difficult for us to accurately view the *saturation* values on the *marks* as well as read the university names. IV-114

CHAPTER V: IMPLEMENTATION
AUTOMATIC VISUALIZATION INTERFACE DESIGNER V-116

Figure V-1: Three components within AVID that correspond to the three stages in the automatic design process: 1) Task interpretation, 2) Visualization design, and 3) Design Realization V-117
Figure V-2: Example task specification V-119
Figure V-3: Task argument structure V-121
Figure V-4: Task class structure V-122
Figure V-5: Flowchart of AVID search strategy. Consists of two main phases: 1) task processing phase and 2) data attribute mapping phase V-124
Figure V-6: Task operators and their corresponding visualization functions V-125
Figure V-7: Partial search tree of house example task V-126
Figure V-8: Example designs generated corresponding to the 6 terminal nodes in the search tree in Figure V-7 ..... V-127
Figure V-9: Visualization design illustrating the different composition types. There is *cluster* composition in the left chart between the *labels* and the *marks*. There is *double-axis* composition in the right chart between the *marks* and the *bars*. There is *single-axis* alignment between elements in the left-chart and those in the right-chart. These composition types were first introduced by Mackinlay [*Mackinlay, 1986a, 1986b*] V-131
Figure V-10: Perceptually inexpressive design of the house task in Figure V-2. This is because *date_on_market* is mapped to the *x-axis* of the left chart and *date_sold* is mapped to *saturation* on the left chart. This makes it difficult to compute the duration on market because there is no perceptual operator for comparing the difference between *positional* and *saturation* values V-135
Figure V-11: Mapping transform designs for house search task on *selling_price, number_of_rooms* and *date_on_market* V-136
Figure V-12: Design with *neighborhood* and *selling_price* mapped to integral properties (*hue* and *saturation*). This makes combined search on both these data attributes easier because only a single emergent property (i.e. *color*) needs to be attended to. V-139
Figure V-13: Mixed task processing methods for the *AND* operator in the house search task. The *date_on_market* condition is pre-computed and mapped to the *x-axis* of the left chart, the *num_rooms* condition is pre-computed and mapped to *hue* on the left chart, however, the *selling_price* condition is performed perceptually by mapping *selling_price* to the *x-axis* on the right chart. V-140
Figure V-14: Using similar task processing methods for the *AND* task V-141
Figure V-15: Example graphical object class specification for interval bar grapheme V-144
Figure V-16: Visualization design with no "objectness" constraint. I.e. it is not possible to associate which house *mark* in the bottom chart corresponds to which house *bar* in the top chart. V-145
Figure V-17: Mapping costs ordered based on data attribute class and graphical property class V-151

APPENDIX A: APPENDIX TO FUNCTIONAL VISUALIZATION TECHNIQUES FRAMEWORK (CHAPTER II) A-164

Figure A-1: Example ODT diagram for the dynamic query slider [*Ahlberg, 1992*] visualization technique A-164
Figure A-2: Highlight object selection and Dynamic query sliders A-165
Figure A-3: P-C composition of selection and Dynamic Query A-165
Figure A-4: Value painting specification A-165
Figure A-5: Combining the object selection and dynamic query techniques using object definition composition A-166
Figure A-6: Combining the object selection and dynamic query techniques using transformation composition A-166
Figure A-7: HomeFinder system specification A-167
Figure A-8: SDM distance operator components: *distance-to* attribute, point of reference, and line of reference A-167
Figure A-9: SDM distance operator specification A-168
Figure A-10: Sequence of images showing different multiplication factors being applied to the A-169
Figure A-11: Changes made to SDM and HomeFinder specifications A-170
Figure A-12: Example house selection technique A-172
Figure A-13: Extended HomeFinder + SDM distance technique A-173
Figure A-14: Effective way of placing 3 constraint points A-173
Figure A-15: Dividing up house concepts into 3 groups based on *selling_price* A-174
Figure A-16: Dividing up house concepts into 3 groups using the *foreach* control operator A-174
Figure A-17: Partition selection A-175
Figure A-18: Using the switch control operator to channel the execution flow A-175

APPENDIX B: APPENDIX TO INSTANTIABLE VISUALIZATION TECHNIQUES FRAMEWORK (CHAPTER III) B-178

Figure B-1: Tweedie's description of the dynamic query and Table Lens techniques B-179
Figure B-2: Data Flow diagram showing relationships between a visualization technique, the user and the visualization designer B-181
Figure B-3: Dynamic query functional specification B-182
Figure B-4: Dynamic query specification with specific object-definition and transformation functions B-182
Figure B-5: Dynamic query specification with input and output types for each object definition and transformation function. The "?" boxes indicate areas where translation functions are needed to convert from one argument type to another B-182
Figure B-6: Dynamic query specification with intermediate functions for inputs and outputs (Note that the visualization function classes are not shown in order to reduce the amount of diagrammatic clutter) B-183
Figure B-7: Dynamic query specification with all inputs required B-183
Figure B-8: Adding a slider input device for specifying the threshold constraint in the dynamic query technique B-184
Figure B-9: Initializing the *min* and *max* properties of the slider input device added in Figure B-8. The *min* and *max* values are derived by computing the *min* and *max* values of the *house-selling-price* data attribute with data transform functions. B-184
Figure B-10: Alternative object definition and transformation functions for the dynamic query slider technique B-186
Figure B-11: Specification for the dynamic query slider technique including object definition, transformation, as well as translation functions B-187
Figure B-12: In this specification we change the translation functions leading into the intersect operator so that the intersect operation is applied to data concepts instead of graphical objects as was the case in Figure B-11. Here, only those data concepts that are both selected by the

slider and that are contained within *related_visualization* can be selected. Note that all the changes made to the specification in Figure B-11 are shown in gray in Figure B-12. ..... B-187
Figure B-13: This specification is similar to Figure B-11 except that here we have included all of the input arguments that are required by the various functions that have yet to be provided. Each missing argument value is indicated with a "?" symbol together with the argument type that is required. In this example there are five missing function arguments..... B-188
Figure B-14: Slider visualization technique with input devices B-189
Figure B-15: One possible instantiation specification of an aggregation technique. This technique allows users to select a set of graphical objects using a *bounding-box*. It then summarizes the underlying data concepts by using a *group-objects* data transform function, thereby creating a new *aggregate-data-type-1* data-type. In addition, one of the data attributes for the selected objects is also chosen for *mean* summarization, and a new *summary-attribute-1* is created to store the mean values. This data attribute can then be mapped to a graphical property. B-191
Figure B-16: Instantiation specification for Visage drag-and-drop technique. This technique allows users to pick a set of graphical objects in one visualization, and then add the underlying data concepts of the selected objects to a different visualization. The *switch* and *case* statements used above are to ensure that the user has indeed selected an origin set of graphical objects and a destination visualization container. B-192
Figure B-17: Instantiation specification for the Table Lens semantic zoom operation . This technique identifies the larger cells in the table (i.e. height > 20, width > 200) and adds a new *text-class* mapping for those larger cells..... B-193

APPENDIX C: APPENDIX TO DESIGN HEURISTICS (CHAPTER IV) C-198

Figure C-1: Cognitive design for the airline-scheduling task (Note that the flights are not all shown here because the table is very large) C-198
Figure C-2: Perceptual design for the airline-scheduling task Each line represents a flight with origin and destination city mapped onto the *y-axis* and arrival and departure time mapped onto the *x-axis*. This is the best design that gets generated when ONLY mapping operations are considered by the automatic system. I.e. this is the best possible design from current state of the art systems. C-201
Figure C-3: Design generated when data processing operations are integrated into the automatic visualization system. The full data set is considered here but data transforms are applied by the automatic system to filter the data set so that only relevant flights are shown. The total downtime before the meeting for the flights from LAX to ORD is shown on the left chart and the total downtime after the meeting for the flights from ORD to BOS is shown on the right chart. C-204
Figure C-4: Solving the airline schedule task with input devices. Here we assume that some of the airline-scheduling task constraints are unknown (i.e. the *layover-city* is not known and the *beginning* and *ending* meeting times are also not known). These constraints can be entered into the system through *sliders* and *option-buttons*. C-205
Figure C-5: Design alternative for airline-scheduling task where flights are filtered based on origin, destination, and layover city information. Arrival and departure times are shown however to allow flexibility in our task meeting time constraint. *Flight_arrival_time* is mapped to the *x-axis* of the left chart and *flight_departure_time* is mapped to the *x-axis* of the right chart..... C-206
Figure C-6: Solving the airline-scheduling task where all searches as well as the total-downtime computations are performed with data transform techniques. *Total_downtime* is pre-computed and mapped to the *x-axis*. Note that in this design there are many more graphical elements than the other airline-scheduling task designs because all possible pairs of flights must be considered and shown. [Note: the indented labels for this design was manually generated] C-207
Figure C-7: This diagram shows how data analysis tasks can be broken down into perceptual operators (mapping transforms) and system computation operators (data transforms) and how

these operators ultimately combine to produce a visualization design (external representation). C-208
Figure C-8: Increasing trend C-212
Figure C-9: Design solutions for total cost computation task (Task C-2). C-219
Figure C-10: Encoding the same data as Figure C-9 but with bars instead of text C-219
Figure C-11: Computing the difference between *out-of-state-tuition* and *room-&-board-costs* C-221
Figure C-12: Computing the ratio between *out-of-state-tuition-costs* and *room-&-board-costs* C-222
Figure C-13: Design solutions for expanded total cost computation task (Task C-4) C-223
Figure C-14: Design solutions for expanded find task (Task C-5) C-224
Figure C-15: Data computation design for Task C-6 In this design the automatic design system is able to utilize the embedding structure of the task to filter the design so that only those universities with *in-state-tuition* less than *room-&-board-costs* are shown. This cuts down on the number of elements that have to be shown significantly, producing a more easily interpretable display. *In-state-tuition* values are then shown on the *x-axis* of the bar chart. C-226
Figure C-16: Design solutions for find task in Task C-7 ..... C-228
Figure C-17: Design alternatives for Task C-8 ..... C-229
Figure C-18: Finding universities where the department with the most faculty is also the department with the most funding. The two mapping alternatives are less effective compared to the data computation solution because in both the mapping designs the *find* task may not be fully pre-attentive while in the data computation design it is very easy to identify the universities with the same top faculty and funding departments. In addition, this information is very well integrated into the design without increasing complexity by much. C-232
Figure C-19: Mapping design where *out-of-state-tuition-cost* is mapped to the blue bar *x-lengths* and *room-&-board-cost* is mapped to the red bar *x-lengths* for a set of universities. In the data set with *Rich-University* there are severe dwarfing problems on the bar lengths making it difficult to accurately estimate cost values. C-234
Figure C-20: Same data set as Figure C-19 but showing the pre-processed difference values between *out-of-state-tuition-cost* and *room-&-board-cost* instead of the original cost figures. Note that in this case it does not matter whether *Rich-university* is included or not, the difference distributions of the two data sets are comparable. I.e. the dwarfing problem is no longer an issue in the data transform design. C-235
Figure C-21: Data computation design for car purchasing task A (Task C-10) The top cars are shown for each car picking category. While this design is very simple and easy to interpret it does not allow us to flexibly take all four car picking attributes into account simultaneously. C-238
Figure C-22: Designs for car purchasing task B (Task C-11). Design (b) is more effective than design (a) because in design (b) it is easy to lookup the score sums and in addition, the individual scores are also given so that we may examine each criteria separately. C-239
Figure C-23: Data computation design for car purchasing task 3 In this design users get the flexibility to enter in filtering thresholds for each of the four car picking attributes through *slider* input devices. The system then pre-computes all cars that fulfill those threshold conditions and then only displays those cars. This produces a much cleaner and effective design compared to the previous designs in Figure C-22. In addition, in this interface the user may weigh each of the four car picking attributes differently (e.g. place more importance on *horsepower* and less on *price*) by setting more or less stringent thresholds. In Figure C-22, each of the four car picking attributes are weighed equally and users are not given the ability to alter this weighting. C-241
Figure C-24: Mapping design-1 for car purchasing task 3 Mapping design that shows the original four stock picking attributes to the user. *Engine-size* is mapped to the *x-axis* of the left chart and *min-price* is mapped to the *saturation*; *city-mpg* is mapped to the *x-axis* of the right chart and *horsepower* is mapped to *saturation*. Desirable cars are those with long (large *engine-size*), unsaturated (low *min-price*) bars in the left chart and short (low *city-mpg*), saturated (high *horsepower*) bars in the right chart. This design is significantly more complex in terms of number of elements and difficulty of interpretation compared to Figure C-23. In addition the *saturation* encodings does not allow for accurate value lookups or comparisons C-243
Figure C-25: Mapping design-2 for car purchasing task 3 Mapping design that shows the original four stock picking attributes to the user. *min-price* is mapped to the *x-axis* of chart 1 (left-

most chart), *city-mpg* is mapped to the *x-axis* of chart 2, *engine-size* is mapped to the *x-axis* of chart 3, and *horsepower* is mapped to the *x-axis* of chart 4 (right-most chart). Desirable cars are those with short bars in chart 1 (low *min-price*), short bars in chart 2 (low *city-mpg*), long bars in chart 3 (large *engine-size*), and long bars in chart 4 (high *horsepower*). This design is more accurate than Figure C-24 because all values are encoded on the *x-axis* (i.e. no *saturation* values are used). However this design is also less integrated and requires significantly more eye movement, display space, and display navigation.....C-244

APPENDIX D: APPENDIX TO IMPLEMENTATION (CHAPTER V) D-246

Figure D-1: An example pair of visualizations that match based on both structure and content. Structurally both designs have two charts, one of which is a bar chart and the other a scatterplot. In terms of content, both designs contain the same data attributes (*object-name, selling_price, neighborhood, owner_salary*) and graphical property classes (*2 positionals,* and *2 retinals*)..... D-246
Figure D-2: Connecting all visualization functions within a node state from innermost task to outermost D-247
Figure D-3: Making task class modifications depending on whether we want to show the *find* task results either through object filtering or by mapping its results to a graphical property..... D-248
Figure D-4: Adding in any necessary translation functions..... D-249
Figure D-5: Adding in input arguments to the visualization functions..... D-250
Figure D-6: Initialization functions for the slider input device..... D-251
Figure D-7: Visualization graphical objects and their corresponding Inventor nodes..... D-252
Figure D-8: Inventor scene graph of visual structure design..... D-254
Figure D-9: Single axis visualization design with and without realizer layout algorithm..... D-255
Figure D-10: Non-unique positional visualization design with and without realizer layout algorithm..... D-255
Figure D-11: An example description of a visualization function..... D-256
Figure D-12: An example description of a connector..... D-257
Figure D-13: An example description of an input device..... D-258
Figure D-14: An example description of a composite visualization function..... D-258
Figure D-15: Functional specification editor..... D-260

APPENDIX E: USING GOMS TO EVALUATE OUR AUTOMATIC DESIGN SYSTEM..... E-262

Figure E-1: Representing search results using text labels with "Y" or "N". Such results can be pre-attentively searched on through pattern matching. E-284
Figure E-2: Representing search results using text labels with "distinguish" or "don't_distinguish". Such results can be pre-attentively searched on through length matching.....E-284
Figure E-3: GOMS estimated total time for Task 1 (search task). The designs are ordered based on increasing cost on the x-axis. The y-axis shows the GOMS estimated total time in msec. All pink bars indicate pure mapping designs (i.e. designs that can be generated with current state of the art automatic design research). All other bars are designs made possible by work outlined in this thesis. E-285
Figure E-4: GOMS estimated total time for Task 2 (computation task). The designs are ordered based on increasing cost on the *x-axis*. The *y-axis* shows the GOMS estimated total time in msec. All pink bars indicate pure mapping designs (i.e. designs that can be generated with current state of the art automatic design research). All other bars are designs made possible by work outlined in this thesis. E-311
Figure E-5: Hybrid design where total benefits are pre-computed for full and associate professors (left chart) but not for assistance professors (right chart) E-312

Figure E-6: Hybrid design where total benefits are pre-computed for only full professors (left chart) and the computation for associate and assistance professors (right chart) as well as the final summation task must be performed perceptually. E-313
Figure E-7: GOMS estimated total time for Task 3 (comparison task). The designs are ordered based on increasing cost on the *x-axis*. The *y-axis* shows the GOMS estimated total time in msec. All pink bars indicate pure mapping designs (i.e. designs that can be generated with current state of the art automatic design research). All other bars are designs made possible by work outlined in this thesis. E-335
Figure E-8: List of universities ranked based on their computed total-benefit values E-337
Figure E-9: Visualization design where the total benefit values are sorted and the sorted rankings are mapped to *x-position* 338
Figure E-10: Visualization where the total benefit values are mapped to bar *x-lengths* 338
Figure E-11: Visualization where the universities are ordered on the *y-axis* based on *total-benefits* and in addition the *total-benefit* values are mapped to *bar-lengths*. 339
Figure E-12: Visualization where the universities are ordered on the *y-axis* based on *total-benefits* and in addition the *total-benefit* values are mapped to *text labels*. 339

APPENDIX F: ENHANCING READABILITY WITH GRAPHICAL & RENDERING TRANSFORMS F-343

Figure F-1: Augmented search algorithm for our automatic design system AVID. Additional steps take into account readability issues and how to solve them F-357

LIST OF TABLES

CHAPTER II: VISUALIZATION TECHNIQUES FRAMEWORK
A FUNCTIONAL DESCRIPTION II-17

Table II-1: Summarization of composition types II-34
Table II-2: Data visualization techniques II-38
Table II-3: Goal categories of data visualization techniques II-39
Table II-4: Mapping visualization techniques II-41
Table II-5: Goal categories of mapping visualization techniques II-42
Table II-6: Graphical visualization techniques II-44
Table II-7: Goal categories of graphical visualization techniques II-45
Table II-8: Rendering visualization techniques II-46
Table II-9: Goal categories of rendering visualization techniques II-47

CHAPTER III: VISUALIZATION TECHNIQUES FRAMEWORK
A CONCRETE INSTANTIABLE DESCRIPTION III-50

Table III-1: List of object definition functions III-59
Table III-2: List of mapping transformation functions III-61
Table III-3: List of data and graphical transformation functions III-67
Table III-4: Input and output translation functions III-70
Table III-5: Input-device Query functions III-72

CHAPTER IV: DESIGN HEURISTICS
DATA COMPUTATION VS. PERCEPTUAL MAPPING IV-82

Table IV-1: Semantic distance for computing the total benefits for associate professors IV-101
Table IV-2: Semantic distances for total benefits task (Task IV-2) IV-104
Table IV-3: Semantic distance for finding average number of teaching staff for a set of universities IV-105
Table IV-4: Semantic distance for an airline-scheduling task which balances total downtime and total cost IV-109
Table IV-5: Semantic distance for finding the total and individual sales IV-111
Table IV-6: Semantic distance for finding a house based on price, size, and distance to workplace IV-113

CHAPTER V: IMPLEMENTATION
AUTOMATIC VISUALIZATION INTERFACE DESIGNER V-116

Table V-1: Task inputs and outputs 120
Table V-2: Summary of actions taken based on task output and embedding status 129
Table V-3: How Information amplifies cognition (from Card et al.[*Card, 1999*]) 133
Table V-4: Data transform constraints for each task class 142
Table V-5: Mapping design constraints for each task class 143
Table V-6: Input devices considered in AVID with their cost properties (* indicate no constraints on an input device property) 147

APPENDIX C: APPENDIX TO DESIGN HEURISTICS (CHAPTER IV) C-198

Table C-1: Tasks and their input and output arguments C-209

APPENDIX E: USING GOMS TO EVALUATE OUR AUTOMATIC DESIGN SYSTEM..... E-262

Table E-1: Summary of all GOMS operators used in the evaluation sequences listed in this appendix E-264

APPENDIX F: ENHANCING READABILITY WITH GRAPHICAL & RENDERING TRANSFORMS F-343

Table F-1: Expressiveness and effectiveness of graphical and rendering transforms with respect to readability. + : Readability issue is supported reasonable well; - : Readability issue is not supported well; *"empty"* : Readability issue is not supported;* indicates the transparency property only..... F-349

LIST OF TASKS

CHAPTER IV: DESIGN HEURISTICS
DATA COMPUTATION VS. PERCEPTUAL MAPPING IV-82

Task IV-1: Airline-scheduling task. The user is trying to find flights to enable a meeting to be held in a layover airport en-route to a destination and to minimize time spent at the layover airport before and after the meeting. IV-82
Task IV-2: Task for determining the total benefits given out to full professors and associate professors. IV-102

APPENDIX C: APPENDIX TO DESIGN HEURISTICS (CHAPTER IV) C-198

Task C-1: Subtask extracted from Casner's airline-scheduling task [*Casner, 1991*] C-214
Task C-2: Computing the total cost for attending a university C-218
Task C-3: Change in task operator from *addition* to *difference* C-220
Task C-4: Expanded university total-cost task C-223
Task C-5: Expanded tuition-cost and room-&-board-cost find task C-224
Task C-6: Change in task embedding structure from Task C-5 (Note that we represent the university set with { ... } here to make the task specification easier to read. However, the data set used is the same as all previous examples in this section) C-226
Task C-7: Find task with a constant argument C-227
Task C-8: Find task with an unknown task argument C-229
Task C-9: Finding the most prosperous department in each university based on funding and faculty size C-231
Task C-10: Car purchasing task A C-237
Task C-11: Car purchasing task B (Note that in this task specification we assume that the min and max values for each data attribute have been pre-calculated) C-238
Task C-12: Car purchasing task C C-241

APPENDIX E: USING GOMS TO EVALUATE OUR AUTOMATIC DESIGN SYSTEM E-262

Task E-1: Search for universities based on the average SAT and ACT scores of attending students as well as graduation rates. E-265
Task E-2: View the total benefits given to faculty for a set of universities. E-287
Task E-3: Sort a set of universities by their total benefits E-314

Chapter I: Introduction

Automatic Visualization Design

Automatic visualization systems have two primary goals: 1) to improve communication between the computer system and users both in terms of effectiveness and breadth and 2) to serve as a design assistant and help facilitate the creation of graphics for information presentation and analysis. To support the more complex and heavy demands that are made of information analysis systems today it is necessary to expand the effectiveness and flexibility with which computers can communicate with users. The range of tasks, data, media types and user preferences that these information systems must accommodate make it unfeasible to anticipate every possible output scenario. I.e. it is impossible to custom design graphics and interfaces here because there are too many possible output alternatives. Automatic visualization systems enable the flexible generation of information presentation graphics that are crafted on a case by case basis to suit the wide range of communication goals that may arise. Design rules and theories of cognition and perception are applied within these automatic systems to ensure the effectiveness and correctness of the graphics generated. Automatic visualization systems can also help users create and design graphic presentations. The SageBrush and SageBook interfaces [*Roth, 1994*] show that automatic visualization research can be applied to help users complete partial designs or browse and adapt existing designs to show new data. This helps users with the more straightforward design operations, leaving them free to quickly explore many more design alternatives. It is important to stress that work in automatic visualization design is not meant to remove the "human" aspect from design and neither is its goal to design a "better" graphic than human designers. Instead, the power of automatic visualization systems is derived from the cooperative process between user and automatic system. The advantage of such systems as a design tool is that it can quickly generate a large range of possible design solutions and show them to the user who can then decide between similarly effective designs based on their preferences. In this way an automatic design system can aid a designer or less experienced user in creating graphics by performing the more mundane and simple visualization design tasks as well as give design suggestions that are based on compiled knowledge from graphic design and perceptual theory. Ultimately, the synergy between the user and the expert design system will be able to generate good designs more effectively and easily.

Previous research in automatic visualization design [*Mackinlay, 1986a, 1986b; Casner 1991; Roth 1994*] focussed on developing rules for mapping data to graphical elements effectively so that the generated designs support the desired user task(s) and can be clearly and correctly interpreted. However mapping data to graphics is only one step in the visualization creation process. Before we map data to graphical

representations it is commonly effective to first process the data either by summarizing, computing, or culling out less relevant elements. By first massaging the data to a more appropriate form before presentation, we can construct more effective graphical designs for expressing user goals. This thesis expands the automatic design process to include these pre-processing data operations. This expansion allows a significant improvement over previous automatic design systems because it enables us to:

1. Generate more effective designs.
2. Address a larger range of tasks.
3. Produce a larger range of interesting design alternatives.
4. Usefully integrate input-devices into the design process.

1. Generate more effective designs

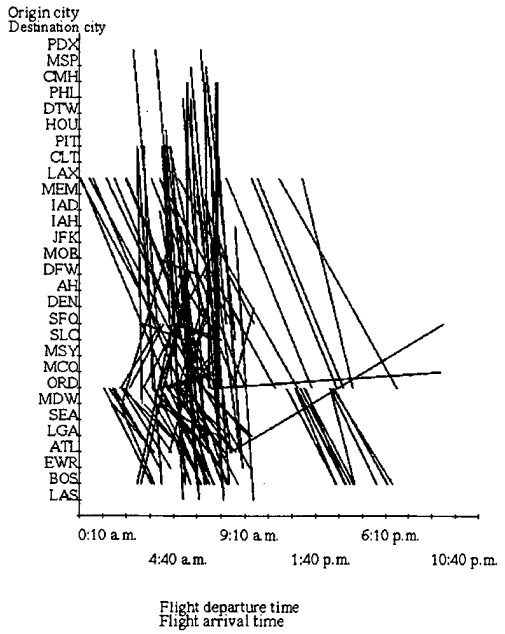
The advantages of this work can be quickly seen in the following airline reservation task that was used in a previous automatic visualization system [*Casner, 1991*]. In this task the user is interested in finding flights from *Los Angeles* to *Boston* with a layover in *Chicago* where they will be a meeting from *2 p.m.* to *4 p.m.*. The user would like to schedule the flights such that the total downtime in *Chicago* before and after the meeting is relatively small.

Flight number	Origin city	Destination city	Flight arrival time	Flight departure time
United_652	ORD	EWR	5:32 a.m.	2:30 a.m.
NW_826	MEM	ATL	10:13 a.m.	8:00 a.m.
Trans_235	MDW	LGA	9:00 a.m.	5:55 a.m.
United_244	LAX	ORD	3:37 p.m.	9:45 a.m.
United_342	ORD	MCO	12:19 a.m.	8:55 a.m.
American_494	ORD	EWR	7:48 a.m.	4:30 a.m.
American_491	BOS	DFW	7:29 a.m.	4:15 a.m.
...				
United_7282	JFK	BOS	8:31 a.m.	7:00 a.m.
NW_5875	MEM	MOB	9:45 a.m.	8:20 a.m.
Delta_725	DFW	SFO	10:54 a.m.	9:00 a.m.
TWA_7762	PHL	JFK	7:53 a.m.	7:04 a.m.
American_346	ORD	LGA	10:04 a.m.	7:00 a.m.
American_340	ORD	LGA	11:36 a.m.	8:30 a.m.
Continental_1943	AH	LAS	9:46 a.m.	8:39 a.m.

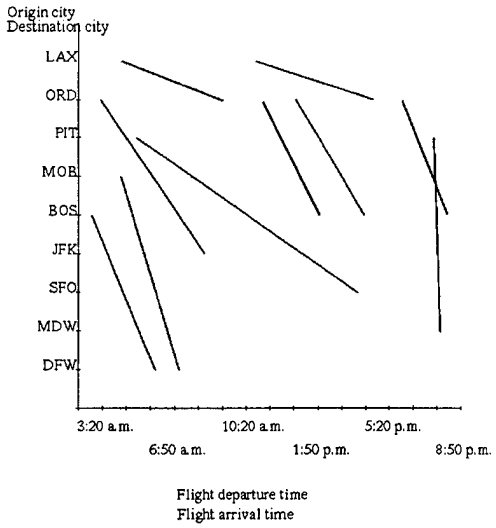
Figure I-1: Cognitive design for the airline-scheduling task
(Note that the flights are not all shown here because the table is very large)

Performing this task cognitively using the raw data arranged in a table format (Figure I-1) would take up to approximately 4 minutes (based on a GOMS evaluation of the visualization). Previous automatic systems explored how this cognitive design can be appropriately mapped to graphics (Figure I-2), which

lowers the total task time to only 30 seconds (assuming no occlusion). In our work we allow automatic preprocessing of data before presentation and the design generated by our system allows the task to be quickly solved in 3 seconds (Figure I-3). A summary of the GOMS estimated time for all three designs is shown in Figure I-4. We briefly describe the perceptual and cognitive steps for these displays in section I-2. The detailed GOMS sequences for these three visualizations can also be found in appendix C-1.



(a) Full data set
This visualization shows all the elements in the data set (i.e. all 135 flights).



(b) Truncated data set.
This example visualization shows the ideal case where there is little occlusion among the different flight lines. This data set was chosen so that it contains some flights that fulfill the task constraints as well as some other random flights that do not occlude one another.

Figure I-2: Perceptual design for the airline-scheduling task
Each line represents a flight with origin and destination city mapped onto the *y-axis* and arrival and departure time mapped onto the *x-axis*. This is the best design that gets generated when ONLY mapping operations are considered by the automatic system. I.e. this is the best possible design from current state of the art systems.

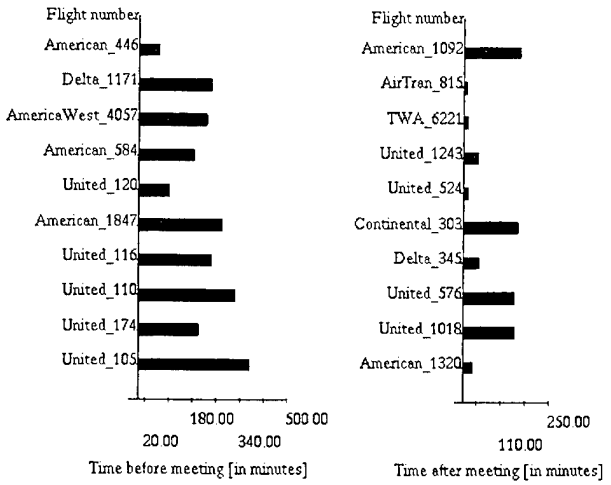


Figure I-3: Design generated when data processing operations are integrated into the automatic visualization system. The full data set is considered here but data transforms are applied by the automatic system to filter the data set so that only relevant flights are shown. The total downtime before the meeting for the flights from LAX to ORD is shown on the left chart and the total downtime after the meeting for the flights from ORD to BOS is shown on the right chart.

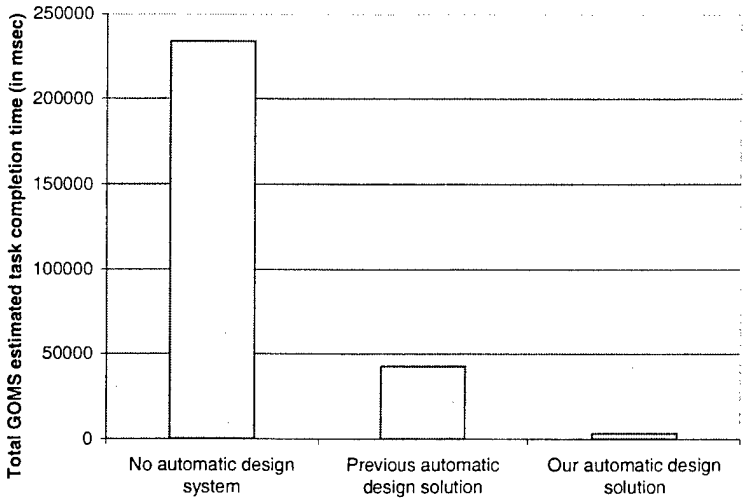


Figure I-4: GOMS estimated total time for solving the airline-scheduling task for a data set of 135 flights. Detailed GOMS sequences for each design are presented in appendix C-1.

2. *Address a larger range of tasks*

Adding data processing functions into the automatic design process also allows us to address a larger range of tasks than was possible with previous systems. Some tasks do not have a purely perceptual solution, including processing of large data sets, abstract mathematical operations (e.g. *log, exp*) or complex calculations that contain multiple related mathematical operations (e.g. *(a+b) / (c+d)*). In this thesis we enable automatic design systems to deal with these problems by automatically computing the

tasks fully or partially so that users are presented with combined computation (data transforms) and perceptual (mapping transforms) solutions. Note that while it is possible to always compute a task fully before conveying it to the automatic designer it is also undesirable and restrictive to do so. As we will show in chapter IV, pre-computing entire tasks may severely constrain the flexibility of users and may not produce the most effective design solution. To generate "good" design solutions we must integrate data processing with mapping decisions because design decisions made in the data processing phase affects the mapping phase and vice versa. And just as we cannot anticipate all combinations of data and information analysis tasks that may be demanded by users, we cannot anticipate all combinations of data processing and mapping operations that are appropriate and useful in our designs.

3. *Produce a larger range of interesting alternative designs*

Expanding automatic design to include data processing operations also allows us to generate a larger range of interesting alternative designs compared to previous systems. Designs generated may be purely perceptual, purely computed, or hybrid computed and perceptual designs. This larger range of choices is important because the data analysis process is an iterative process where users first construct mental models of their current tasks and based on these mental models, pose design requests to an automatic system. Depending on the results of the design request, users may then update their mental models and then repeat the process. The ability of users to arrive at useful answer(s) to their data analysis problems depends on the range and quality of design solutions returned by the automatic system as this will facilitate the next iterative cycle. The wider range of design alternatives provided by our system enable users to better match design solutions to their data analysis goals as well as personal preferences.

4. *Usefully integrate the application of input-devices into the design process*

Input-devices are very effective for allowing end-users to flexibly change a visualization design interactively. This allows large data spaces to be represented because we can interactively focus in on different subsets of data elements at different times. Previous automatic systems did not consider the use of input-devices. This is primarily because these systems always showed all of the available data (i.e. there was no data summarization, computation, or culling). As a result there was no need for users to navigate through the visual representation by using input-devices. Unfortunately, the lack of interactivity and data summarization operations also constrained these systems so that they can only address problems with relatively small data sets (< 20 elements). This was clearly illustrated in Figure I-2b where showing the entire data set of 135 flights produced a visual display that was too occluded to be of any use. In this thesis we add input-devices and data transforms into the automatic design process so that our system can deal with larger data sets (> 100 elements) through data culling, summarization, and interactivity.

To integrate data processing functions into automatic design we develop three core technologies in this thesis:

1. *Methods*: We develop a way to characterize data and mapping functions within a visualization design, how these methods may be combined with each other, with the output media and with available input-devices.
2. *Principles*: We also develop a set of design dimensions and strategies that can help us gauge the quality of different design alternatives. These dimensions and strategies determine when and how data and mapping methods should be used based on user tasks, data, and preferences.
3. *Systems*: We show that the methods and principles developed are complete and applicable by using them to implement an automatic visualization design system. We then evaluate the system through a series of GOMS analyses to show that the results generated by our prototype designer are correct (i.e. they support the input tasks) and are ordered based on cognitive, perceptual, and motoric complexity. I.e., the most effective or least complex design is generated first and the most complex design is generated last.

I-1 Methods: Visualization Techniques Framework

In this thesis we develop a framework that characterizes the function and structure of visualization techniques that are used to create and modify visual designs. This framework provides our automatic design system with the necessary constructs to build visualization interfaces that may contain data processing functions, mapping functions, as well as input-devices.

Each visualization technique within our framework is defined to have a selection component and a transformation component. Selection can be achieved through enumeration or through a constraint function (functional description). Transformation can be achieved using the four different functions within the visualization creation process shown in Figure I-5. To build richer visualization techniques, we can combine multiple techniques together through a set of composition functions.

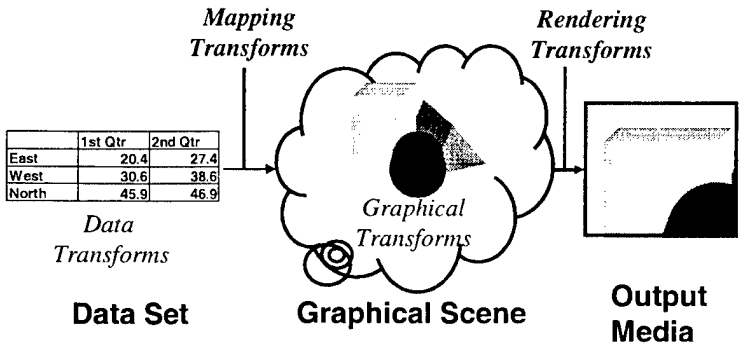


Figure I-5: The four phases of the visualization creation process

The visualization creation process consists of four primary phases: data, mapping, graphical, and rendering. Initially in the *data* phase the task data is processed and a portion of it is chosen for subsequent mapping. In the *mapping* phase the chosen data elements from the previous phase are mapped onto *graphical properties* (e.g. *color, position, shape*) and *graphical objects* (e.g. *marks, bars, lines*). In the *graphical* phase, the graphical scene constructed from data mapping is further processed to accommodate changes that may not be reflected in the data set. For example objects in dense areas may be made smaller to avoid occlusion. Finally in the *rendering* phase the graphical scene is transferred onto an output media. There are currently many different media types available (e.g. PalmPilot™, CRT screens, image projection screens) with different constraints on visualization size, number of colors, resolution, mobility, etc. All these constraints affect the way with which the visualization design may be displayed and explored.

Previous automatic design systems only considered the use of mapping functions. In this thesis we expand automatic visualization design to include both data and mapping functions. Even though our automatic design system only uses functions from the data and mapping phases we decided to lay out all four transformation classes in Figure I-5 in our framework because it helps us better understand the roles that data and mapping functions can play in the design process, it allows graphical and rendering functions to be easily integrated into the automatic design process in the future, and it increases the applicability of our framework, allowing us to categorize and analyze current visualization systems and techniques.

Below we show how a very simple visualization technique can be specified based on the primitives and composition rules in our framework. The technique is a simple highlighting technique that allows users to select a set of objects using a *bounding-box* and then subsequently highlights the selected objects *red.*

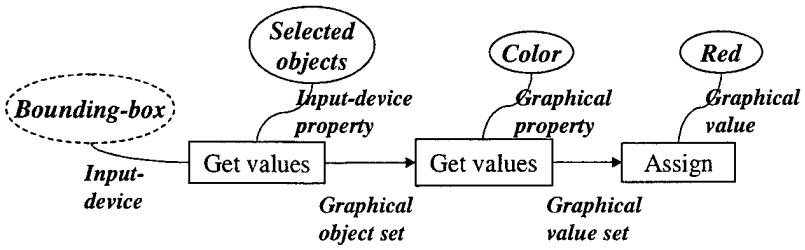


Figure I-6: Highlight technique specification. The input-device bounding-box is used to select a set of objects. These objects (*selected-objects*) are extracted from the bounding-box device using the *get-values* function. We then get the *color* values from all of the selected objects using a subsequent *get-values* function. Finally we change all of the *color* graphical values to *red* using the *assign* function.

All function primitives are shown with normal *Times-Roman* font within rectangles and all inputs to the primitive functions are shown as ***italicized bold*** text within ovals. Inputs provided by users are shown with dotted ovals and those provided as designer defaults are shown with regular unbroken ovals. The directed arrows (→) connecting one primitive function to another indicate a flow of objects or values from

a source function to a destination function. The ***italicized bold*** labels next to the connecting links indicate the types of objects or values that are being passed through that link.

Aside from being a crucial component to our automatic design system, this framework also provides multiple other contributions to the field:

1. *Prototype and Tailor Visualization Interfaces*

Our framework helps designers prototype and tailor visualization interfaces. For example, a visualization designer can very easily adjust the technique in Figure I-6 so that it highlights objects *blue* instead of *red* by changing the oval marked *red* to *blue*. Alternatively we can let end-users specify the highlight color through an input-device by replacing the oval marked *red* with an input-device. The high-level visualization techniques description language provided by our framework enables designers to create and adjust visualization techniques without resorting to writing code.

We show more examples of our visualization techniques language and how they can be created and varied in chapters II and III. For now however, it is important to stress that this visualization language is not meant for end-use. A user-friendly interface should be built on top of the specification language before it can be readily accessed by end users. For example the SAGE system [*Roth, 1994*] has an underlying language for describing the data and graphical elements within a visualization as well as the mapping relationships among them. However it is also attached to a graphical user interface, SageBrush, that provides end users with simple drag and drop techniques for utilizing this language. The same situation applies for the data, mapping, graphical, and rendering functions considered in this work. A friendlier interface is needed for end-users but this interface must be based on an underlying language that captures the functionality and structure of visualization techniques. Our framework can serve as this basis.

2. *New Two-level Design Methodology*

Our framework also presents a new design methodology for creating visualization techniques. This methodology divides the design process into two different levels of abstraction: a *functional* level, and an *instantiation* level. At the *functional* level designers focus on providing users with the proper operations to serve their current goals. I.e. focus is on choosing appropriate functions from each of the four visualization phases and combining these functions. The *instantiation* level, on the other hand, is more concerned with the general appearance and usage of the visualization technique. At this level focus is on choosing appropriate devices for input entry, choosing effective graphical attributes for visual feedback, as well as general layout of the visualization interface. This division helps designers separate the two different aspects of visualization techniques, *function* and *form*, so as to decrease the likelihood of falsely constraining functionality based on appearance concerns. This two level methodology is an advance over previous work

that only considers either the *functional* [*Tweedie, 1997; Card, 1999*] or *instantiation* [*Brodlie, 1991*] levels in isolation.

3. Exploration of the Visualization Techniques Design Space

Finally the framework helps to scope out a large part of the visualization techniques design space, and allows for more systematic exploration within that space. We show at the end of chapter II the space of current visualization techniques and how they may be combined to form new methods. A description of the current space of techniques is important because it shows us the areas we have explored and points to new and future areas of exploration. For example we found that most visualization techniques that are used to search for data objects utilize simple feedback methods to show their results. Feedback for these techniques usually involve changing a single graphical property (e.g. *color*) to different constants (e.g. *red*). Thus one new area of exploration could be in developing useful object search techniques with richer feedback methods that change multiple graphical properties simultaneously in meaningful ways.

I-2 Principles: Design Dimensions and Strategies for Measuring the Goodness of Visualization Designs

The visualization techniques framework described in the previous section provides an automatic design system with the proper language for expressing a wide variety of visualization designs. However there are many possible different alternative designs that fulfill a particular data analysis task. For example, the airline-scheduling task presented earlier can be solved using any of the three alternative designs shown in Figure I-1, Figure I-2 and Figure I-3. Thus in addition to a visualization techniques language, an automatic system must also be equipped with design rules and strategies that help guide it down more promising design paths and prevent it from generating ineffective designs. For this purpose we develop a set of design dimensions for measuring the goodness of different design alternatives as well as a set of design strategies that help our system first generate designs that are deemed more effective based on our design dimensions.

Our design dimensions are built upon previous work by Abowd and Beale for measuring the effectiveness of user interfaces. This framework calculates the overall "goodness" of a visualization design or its "*semantic distance*" by using four distances: *articulatory distance, functional distance, expressive distance,* and *observational distance*. *Semantic distance* refers to the degree with which user goals are fulfilled by the visualization. A large semantic distance means that the goals are not achieved well and a small semantic distance means that the goals have been satisfied acceptably. *Articulatory distance* measures the amount of input-device manipulation required from users. *Functional distance* refers to whether the system possesses software functions or procedures capable of achieving user tasks. *Expressive*

distance determines whether sufficient feedback or information is provided to users to solve the input tasks. Finally, *observational distance* refers to the ease with which a user can interpret system feedback. Specifically, *observational* distance measures the effectiveness of the visual objects, visual properties, and visual compositions used to fulfill the input analysis tasks. Based on these dimensions we develop a set of design strategies that help minimize the *overall semantic distance* of a visualization.

The data processing operations added by this work can improve the *semantic distance* of a visualization design by offloading difficult cognitive operations onto the computer system in addition to offloading them onto the user's perceptual system with mapping transforms as was done previously. For example consider the airline-scheduling task presented at the start of this chapter. The pure mapping design (Figure I-2) encodes each flight with a *line* graphical object. The *origin* and *destination* cities are mapped to the *y-axis* and the *arrival* and *departure* times are mapped to the *x-axis*. To perform the task, users must first search for all lines that originate from *Los Angeles* (LAX) and flies to *Chicago* (ORD) as well as originate from *Chicago* (ORD) and flies to *Boston* (BOS). Next the set must also be narrowed down to only those flights that arrive before the *2 p.m.* meeting time in *Chicago* and leaves after *4 p.m.* (i.e. end-point of LAX-ORD flight is to the left of *2 p.m.* on the *x-axis* and starting-point of ORD-BOS flight is to the right of *4 p.m.*). In contrast, Figure I-3 uses data transforms to offload these cognitive search tasks to the computer system instead of to the perceptual system. Specifically, the computer system performs the *city* and *time* search and only presents those flights that fulfill both the city and time constraints in the task. As a result the design is less cluttered and easier to interpret compared to the pure mapping design (Figure I-2).

However, a significant portion of the information from the original data set has been filtered out in Figure I-3 so that if we changed our meeting time or our meeting venue the data transform design would no longer be usable and we would have to generate a new visualization. In contrast the pure mapping design (Figure I-2) is more flexible and can better accommodate changes in user goals (i.e. the mapping design can still be used to solve the modified scheduling task). Thus depending on the demands of current tasks, an automatic design system may choose to apply different blends of data and mapping transforms. In chapter III we explore these issues and develop design strategies that can help our automatic system decide when it is more appropriate to use data transforms to offload a task onto the computer system and when it is more appropriate to use mapping transforms to offload a task onto the user's perceptual system.

In appendix F we outline how graphical and rendering functions can also be integrated into automatic design in the future. Specifically, graphical and rendering functions improve the semantic distance of a visualization design by addressing readability issues. By readability we mean problems arising from constraints of the output media and its interactions with our perceptual system that impede the optimal use of a visual design (e.g. *object occlusion*). It is crucial to address these readability issues because they may cause an otherwise valid design to become unusable because of extreme clutter, or overly small graphical

representations. In appendix F we identify four important readability issues: *occlusion, density, dwarfing* and *information proximity* and discuss how these issues can be addressed through the use of graphical and rendering functions. Readability problems can sometimes also be avoided through judicious use of data transform functions or by mapping the data to a larger graphical representation. We discuss some of these data and mapping readability enhancements in chapter IV. Previous automatic systems did not consider readability issues because it is difficult to address these issues with only mapping transform functions..

I-3 Systems: AVID – Automatic Visualization Interface Designer

Finally to show that our theoretical concepts are sound, practical, and sufficiently complete, we implement an automatic design system, called AVID, based on our framework as well as our design dimensions and strategies. AVID accepts a task specification like the one shown in Figure I-7 as input. Based on this task specification it will generate a series of design alternatives ranked based on their effectiveness with respect to the input task(s).

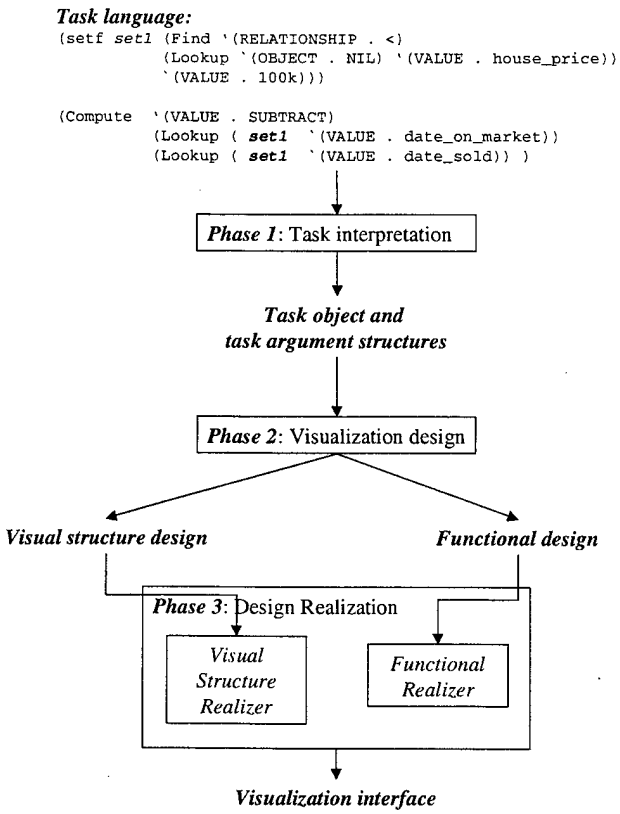


Figure I-7: Three components within AVID

AVID, consists of three components corresponding to the three stages of the automatic design process (Figure I-7):

1. The task interpretation phase

Initially, a higher level agent (user or a domain specific system) that has a deeper understanding of the problem domain generates a set of tasks for AVID. Tasks are expressed using a simple language based on the EDA (Exploratory Data Analysis) task model first developed by Tukey [*Tuket, 1977*] and later refined by Casner [*Casner, 1991*] for automatic design. This language is relatively low-level and its purpose is to capture important components of a task that may affect the visual design process. We do not expect typical end users to specify tasks in this language; rather, specifications will most likely be generated by domain specific systems that use graphics to present and summarize their results to users, such as automatic planning systems, automatic information analysis systems, agent based information gatherers, etc. The task interpreter within AVID evaluates the input task language and generates a set of task objects and argument structures.

2. The design phase

AVID's design component parses the task objects and argument structures generated from the task interpretation phase and converts them to design constraints and cost preferences. These design constraints and cost preferences are generated based on the design dimensions and strategies we discussed in section I-2 (detailed descriptions are in chapter IV). Based on these constraints, AVID explores the design space for the input tasks and automatically generates a set of visualizations ordered from best to worst. These output designs are expressed in a language that captures the visual structure of a visualization interface as well as any underlying transform functions and active interactive components. Visual structure descriptions have been developed in previous work [*Mackinlay 1986a, 1986b; Roth, 1990*]. As was discussed in section I-1, this thesis develops a language for capturing the functions and active components within a visualization (detailed descriptions are in chapters II and III).

3. The realization phase

AVID's "realizer" component interprets design specifications generated by the design component and renders an active visualization interface. This component makes layout decisions and assigns default values to visual components that are left unspecified or unconstrained in the design specifications. Currently, AVID's realizer is capable of interpreting most of the visualization technique primitives described in this thesis (e.g. computations, set-operations, threshold operations, etc). By combining these primitives it can generate a wide range of interactive behaviors such as aggregation, painting, dynamic queries, simple semantic zoom, SDM graphical manipulation operations, navigation operations, etc.

In chapter V we provide details on how our visualization techniques framework as well as design dimensions and strategies are codified within our automatic design system, AVID, and how the design search space is explored. In appendix E we perform a series of GOMS evaluations on the designs generated

by our automatic system to ensure that its design rankings conform to cognitive, perceptual, and motoric complexity.

I-4 Previous Work

This thesis builds upon a wide variety of previous work. In the following sections we divide work related to this thesis into two classes: 1) work that pertains to the visualization techniques framework, and 2) work that pertains to automatic visualization design (including design dimensions and strategies, as well as system implementations).

I-4.1 Visualization Techniques Framework

There are two classes of visualization frameworks: *functional* frameworks and *instantiation* frameworks. Some example *functional* frameworks include Tweedie's DIVA research [*Tweedie, 1997*] and Card et al's framework [*Card, 1997, 1999*]. These frameworks are high-level and are used to analyze and classify existing techniques based on task, data, functionality, etc. Instantiation frameworks such as Data Explorer, IRIS Explorer, and AVS [*Brodlie, 1991*], on the other hand, establishes a concrete language for describing visualization techniques. Instantiation languages are very detailed and describe visualization techniques completely. Because they are much lower-level compared to functional languages, they are also less appropriate for the analyses and classification of techniques. However, instantiation descriptions, unlike functional descriptions, are realizable or renderable (i.e. these descriptions can be easily translated into an active visualization interface). Our framework differs from all previous frameworks because it encapsulates both the functional and instantiation levels of descriptions. In appendix B-1 we compare our framework to previous work.

The design of our framework is based on previous work in visual specification languages [*Jacob, 1986*], user interface languages [*Foley, 1990*; *Card, 1990; Mackinlay, 1990*] and visualization frameworks [*Card, 1997, 1999*].

I-4.2 Automatic Visualization Design

There are hundreds of rules that graphic designers use to generate visualizations based on their intended task and the data they represent. In addition, there are also a large number of rules that can be derived from psychophysical literature and from user testing of visualization systems. In the next sections we present some background on automatic visualization systems and their internal heuristics which primarily consists of graphic design rules. In addition we will briefly describe some of the work performed in perceptual theory and visualization system testing that can also be used to support automatic visualization design.

I-4.2.1 Automatic Visualization Systems

Tufte [*Tufte, 1983*] and Bertin [*Bertin, 1983*] started the initial work in laying out a set of useful graphic design rules and in characterizing the structure of visualization displays. In his book, *A Semiology of Data Graphics*[*Bertin, 1983*], Bertin identified some of the most important issues in visualization design and exposed many of the important artifacts in their structure. Bertin's work was later refined by Mackinlay [*Mackinlay, 1986a, 1986b*] who developed a syntax for expressing the components of visualizations, effectiveness criteria to decide when and how to use the different graphical components and rules of composition that specify how and under what conditions graphical elements can be combined. Mackinlay then used the expressiveness and effectiveness criteria that he developed to implement a system called APT that could automatically design a well-defined set of visualizations. Casner [*Casner, 1991*] later continued Mackinlay's work by taking a task centered approach to creating visualizations. While Mackinlay previously generated visualization designs solely based on the structure of the input data, Casner now also considered user goals. Casner proposed decomposing a user task into a series of logical operators. These logical operators were then replaced by more efficient perceptual operators where possible, and visualizations were then created based on these perceptual operators. The SAGE system [*Roth, 1990, 1994*] carried this area farther by developing a richer data representation for visualizations thus allowing a wider and more complex set of abstract visualizations to be generated. The SAGE system also allowed users to direct the automatic design system by entering in design preferences in the form of partial designs or a previous favorite design.

Our work builds on these previous systems in two primary areas. First of all, these previous systems only considered the issue of how data can be effectively mapped to graphics. In our work, we additionally consider how the input data can be effectively processed before it is mapped and shown to users. Secondly, these previous systems only generated static, non-manipulable visualizations. Our work allows automatic design systems to generate interactive visualization interfaces so that users can navigate through the visual representations and explore larger data sets than was previously possible.

I-4.2.2 Psychophysical Studies

Most of the previous work done on developing effectiveness and expressiveness criteria for automatic design are based more on well established graphic design rules [*Tufte, 1983, Bertin, 1983*] rather than on perceptual theory. This is because it is difficult to abstract from the low-level results contained in perceptual literature and apply them to higher level perceptual operations that occur in visualization analyses. A large cause of this complexity is due to the presence of a wide variety of graphical styles and graphical properties that may be used. This makes it difficult to isolate the effects of each element and even more difficult to determine the conflicts and relationships between the different graphical artifacts. Studies of perceptual theory are further complicated by the external knowledge of graph reading that is assumed of the user.

Thus it is not feasible to define effectiveness and expressiveness criteria solely based on psychophysical results. Doing so will produce an incomplete model of design and limit the generality of the system. However that is not to say that psychophysical results cannot be used. Green [*Green*] showed various instances in which psychophysical literature can be used to support some of Bertin's design strategies and other instances in which the literature showed errors in those strategies. Senay and Ignatius [*Senay, 1994*] are also beginning to apply psychophysical results to design decisions. In this work we will also use perceptual theory [*Livingstone, 1988*; *Treisman, 1982, 1988*] to enrich our design strategies.

I-4.2.3 User Studies on Visualization Interfaces

There have been many user tests conducted to show the effectiveness of new interactions and visual techniques [*Hollands, 1989*; *Ahlberg, 1992; Plaisant, 1996*]. These tests are usually conducted over a small set of specific tasks and are used to illustrate the usefulness of newly introduced techniques and visual representations. While such experiments are effective for demonstrating the utility of new ideas, they are usually not broad or general enough for us to derive general design rules and strategies.

I-5 Summary

The main contribution of our work is in adding data processing decisions into the automatic visualization design process. This is in contrast to previous work in automatic design that only considered mapping transforms. Our work expands the quality and breadth of designs that may be generated and allows automatic systems to address a larger range of tasks as well as larger data sets. In addition, our work also expands automatic design systems so that they may now begin to generate interactive interfaces. All of this enhances human computer communication because a greater, improved, visual vocabulary allows richer and more complex concepts to be conveyed. In addition, the effectiveness of AVID as a design assistant is also increased because it is able to provide a larger range of "good" design alternatives and choices to users. Our design system also culls out bad designs (i.e. task inexpressive designs or designs that do not support the input task(s)) as well as duplicate designs. This saves users from having to devote attention to these less appropriate visual representations while still having good coverage of the design space.

In order to integrate data processing operations into automatic design, we developed three technologies: 1) a visualization techniques framework, 2) design dimensions and strategies for measuring the goodness of various visualization designs, and 3) an automatic design system (AVID) that is able to automatically design visualizations based on a set of user input goals. Our visualization techniques framework provides our design system with a set of primitives and composition rules from which it may build and design visualization techniques. In addition to being a crucial component in our automatic design work, our framework also stands as a contribution in its own right. First of all the framework simplifies the creation and prototyping of visualization techniques by providing designers with a higher level API set.

Secondly the framework provides a new design methodology that separates the design process into two levels, functional and instantiation, and through these two levels promotes better functional design of techniques. Finally our framework allows a designer to systematically explore the visualization techniques design space and identify design holes within that space.

In addition to the framework, we also developed a set of design dimensions and strategies that help our automatic design system pick the best or most effective design alternatives for the current task(s). These dimensions and strategies can also be applied by human designers as a quick evaluation of their designs and as yardsticks of comparison among multiple current designs. Finally we implement an automatic design system based on our framework and design strategies. This automatic system shows that our theories are sound and complete enough to be actualized.

Concurrent research is also underway for combining the graphics generated by automatic visualization systems with text [*Kerpedjiev, 1997*]. Research in this area, while related, does not deal with the same issues that are relevant to automatic visualization design. Rather, work in this area assumes the existence of automatic text and visualization generation systems, and focuses instead on how best to integrate these two communication media. Therefore, the advancements made by our research to automatic visualization design will naturally feed into the combined text and graphical work as well.

I-6 Walkthrough

This thesis is divided based on the three main technologies presented above. Chapters II and III describes our visualization techniques framework including primitives, composition rules and how new techniques may be created by combining previous methods. Chapter IV contains a set of design dimensions for measuring the effectiveness of visualization designs as well as a set of guidelines that discuss when it is appropriate to use data transforms and mapping transforms. Chapter V shows how we integrated our framework and design rules into an automatic design system. Chapter VI presents some concluding thoughts on the work, discusses its scope, and presents a summary on its impact and how it can be expanded in future work. The first 4 appendix sections (appendix A, B, C, and D) are organized to provide additional information and examples on the material in chapters II, II, IV, and V respectively. Appendix E presents a series of GOMS evaluations on the designs generated by our automatic system and shows that our design output does indeed conform to cognitive, perceptual and motoric complexity. Finally appendix F discusses how we anticipate readability issues can be addressed using graphical and rendering functions.

Chapter II:
Visualization Techniques Framework

A Functional Description

The goal of this thesis is to integrate data processing decisions into the design process of an automatic visualization system. This work enhances the quality and breadth of visualization designs that can be automatically generated as well as expands the range of tasks that can be addressed by an automatic system. In order to integrate data processing operations into automatic design, we must first understand what data processing operations are available, how they can be applied to data elements within a visualization, and how they may be combined together with data-to-graphical mapping operations[1]. To achieve this, we analyze existing visualization systems, and develop a framework or layer of abstraction for understanding current visualization techniques, the types of functions they are composed of (including data and mapping functions), as well as how they are built, combined, and used.

Creating this framework, however, is a difficult task. The widespread development of new visualization techniques in recent years, due to significant increases in information processing demands, have left them fragmented, making it difficult and expensive to combine, customize, or generalize their functionality. Visualization systems are often written for a variety of domains and exist at many different levels of granularity. In addition, they provide a wide range of functions that operate on such disparate objects as inputs devices (*scroll-list, bounding-box*), data concepts (*houses, people*), data attributes (*selling_price, num_rooms*), graphical objects (*marks, interval-bars*) and graphical properties (*color, shape*). Techniques that appear to be physically identical may share very little functional similarity and vice versa.

In this chapter and the next we present a visualization framework that models the functional operations (including data and mapping operations) within various visualization techniques as well as the relationship of these

---

[1] Mapping operations capture how data elements can be mapped to graphical elements, so that complex cognitive processing tasks can be offloaded onto our perceptual system. Previous automatic systems only considered the use of mapping operations. This thesis expands automatic design to include both mapping and data functions.

functions to other visualization elements such as input-devices, data concepts and graphical objects. This framework allows us to effectively create and customize visualization techniques as well as enables us to integrate a large set of powerful functions into our automatic visualization design system, thereby increasing its communicative and design effectiveness. Note that rather than only capturing data and mapping operations, as is needed by our automatic designer, we decided to establish a wider framework that covers all visualization design functions (i.e. including data, mapping, graphical and rendering functions). This broader framework is flexible, and provides us with a better understanding of the role that data processing transforms may play in design, not only with mapping functions, but also with graphical and rendering operations. This broader framework is also easily extensible so that graphical and rendering visualization designs can be integrated into automatic systems in the future.

Our visualization techniques framework stands as a contribution of its own and can be used to characterize and capture the state of visualization techniques today. Contributions of our framework include:

1. *Visualization function primitives*: Our framework presents a set of primitive functions that commonly occur in visualization techniques. These primitives form the basic building blocks of our automatic design system. In addition they give us a better understanding of the class of tasks that can be achieved by different visualization methods and allow us to consider such methods at the same level of granularity (by decomposing them down into the same set of primitives). We show in section II-3 that this set of primitives can express a wide range of visualization methods. These functional primitives can also provide the basis for establishing a visualization techniques library which will simplify the process of creating interactive visualization systems.

2. *Composition rules for merging visualization primitives and exploring the design space*: Once we have defined a set of primitives, we specify rules that determine how these primitives can be combined to form more complex behaviors. These composition rules are very powerful because they allow us to generate an infinite set of visualization techniques from a small set of primitives. By combining together components of existing visualization methods, we can adapt these methods to serve in new domains, devise interesting new ways of achieving tasks, and begin exploring and expanding the design space of visualization techniques.

3. *Visualization independent specification of visualization techniques*: Our framework provides a general language for specifying visualization techniques that is not tied to any particular visual representation. Once specified, a visualization technique may be easily attached to a variety of visualization designs. Such flexibility increases the effectiveness with which we are able to generate, prototype and test visualization techniques.

4. *New design methodology for analyzing the interactive design space*: Our framework presents a two-level design methodology for creating visualization techniques: the *functional level* and the *instantiation level*. The functional level is a more abstract level of characterization that allows us to group, categorize, and reason about techniques based on their functionality and application to tasks. The instantiation level, on the other hand, characterizes techniques based on a set of low-level primitives. At this level we capture all the specifics within a

technique so that based on the instantiation description we can fully generate a visualization interface. Because the instantiation description is detailed and low-level, it is difficult to make generalizations about the various visualization techniques, unlike in the functional level. This low-level description, however, is necessary for our automatic design system because it must be able to describe and generate instantiable or realizable designs. In addition, primitives at the instantiation level form a useful visualization API set. Our framework describes both the functional and instantiation levels as well as presents a systematic process of how to move from a functional description into an instantiable description. This is an advance over previous frameworks that only considered either one of these levels in isolation. Refer to appendix B-1 for a more complete discussion of the differences between our framework and previous work.

In this chapter we describe the functional level of our design methodology. In particular we show how visualization techniques can be functionally decomposed into two primary components, *object definition* and *transformation*, as well as how these two-component techniques can be combined to create interesting behaviors. Our automatic design system later uses this object definition/transformation framework (*ODT framework*) to build and generate visualization designs that utilize both data processing and mapping functions. To illustrate the generality and applicability of our framework, we will also show how it can be used to map out part of the visualization techniques design space, and give some interesting observations made from analyzing that space. Readers who are only interested in the automatic design aspects of this thesis can skip section II-3 of this chapter as it pertains to the generality and scope of the framework rather than to its use in our automatic system. By using our framework to explore current and future techniques, however, we show that it is not only useful to our central thesis in automatic design but also generally applicable to the analysis of a large range of techniques (some of which are not currently captured by our automatic design system). The next chapter will explore visualization design at the concrete instantiation level, as well as evaluate the entire framework based on completeness, coverage, and practicality.

II-1 Visualizations & Visualization Techniques

A *visualization* is a graphical rendering of a set of data attributes. The process of creating a visualization begins with the data transformation phase as is shown in Figure II-1. *Data transforms* are used to calculate derived results or summarize attribute values within a data set. For example we can use a *subtraction* data transform to compute the duration that a house stays on the market from the *date_on_market* and *date_sold* house attributes. Alternatively we can use the *mean* data transform to summarize the selling price of all houses in the *Shadyside* area. Data transforms can also be used to compute new meta-data from an existing data set such as the number of times a particular object appears, or the alphabetical or numerical ordering of a set of values. Data transforms exist solely in the data realm and are used to generate new data concepts and values based on the existing data set.

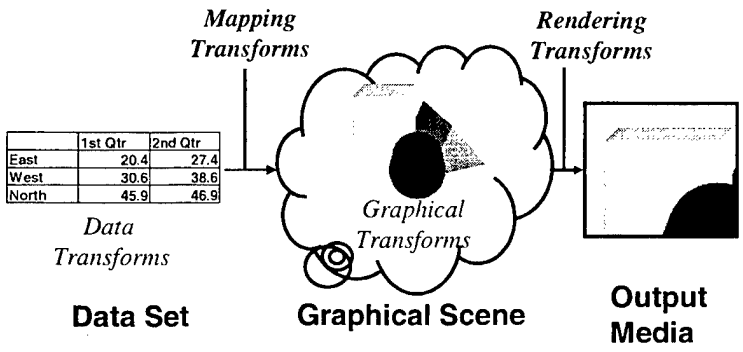
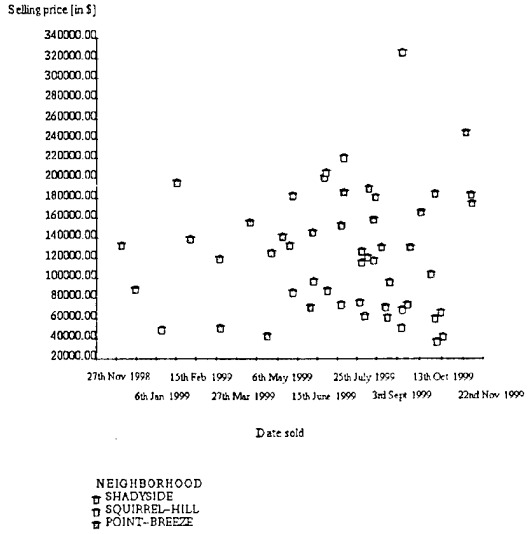
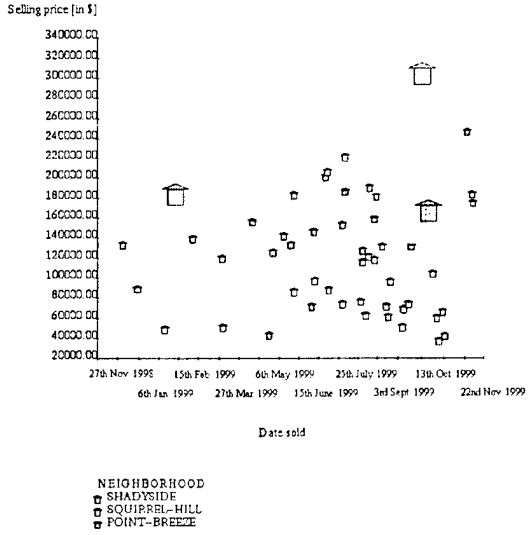


Figure II-1: Visualization generation process consisting of four transformation classes (data, mapping, graphical, and rendering) across three different realms (data, graphical, output media)

Once we have processed all the necessary data, we proceed to the mapping phase where data concepts are mapped to graphical objects and data attributes are mapped onto graphical properties. For example, the visualization in Figure II-2a contains four mapping transforms. An object mapping transform represents all house concepts with graphical *marks*, and a set of attribute mappings link different data attributes of the house data concepts such as *selling_price, neighborhood* and *date_sold* to different graphical properties of the *mark* graphical objects such as *x-position, color,* and *y-position* respectively. At the end of the mapping phase we would have constructed a *graphical scene* representing the data we want to show in the visualization. A *graphical scene* is an abstract model of a boundless space, capturing the position, relationships and appearance of all visual objects (e.g. *marks, axes, legend, labels*) within a visualization.



(a) Several mapping transforms are used to show the *house* data concepts and some of its attributes



(b) Same visualization as Figure II-2 but several house objects have been enlarged with a graphical transform

Figure II-2: Example visualization with house data. Each mark represents a *house* data concept. The *x-axis* shows *date_sold*; the *y-axis* shows *selling_price*; and *color* shows *neighborhood*.

Graphical transforms are used to change the appearance of objects within a graphical scene. For example in Figure II-2b graphical transforms are applied to several selected or focus graphical objects from Figure II-2a so that they appear larger and more salient than the other objects in the visualization. Note that these *size* enlargements do not correspond to any information in the data set and thus cannot be appropriately shown with mapping transforms. Graphical transforms can also be used to change other graphical properties (e.g. *color, shape, position*) and other graphical object classes such as *legends*, *axes* or even the entire *chart region*.

Finally in the rendering process, abstract graphical objects in the graphical scene are transferred onto a bounded output media such as paper, a physical 3D model, or a CRT screen. Figure II-2 shows a rendering of a visualization design on paper. Different media types constrain the classes of techniques that can be used as well as their effectiveness. In this thesis we will only consider the use of CRT screens, thus in our work rendering transforms describe for each screen pixel, the part of the graphical scene to which it corresponds. Note that rendering transforms are the only transform class that operates on physical screen space. All other transformation classes operate on abstract objects such as data concepts and data attributes or graphical objects and graphical properties. The visualization generation process presented here is based on previous work by Card et al.[*Card, 1999*] and Chuah & Roth [*Chuah, 1996*]. These four transform classes in the visualization generation process (Figure II-1) form the basis of visualization techniques.

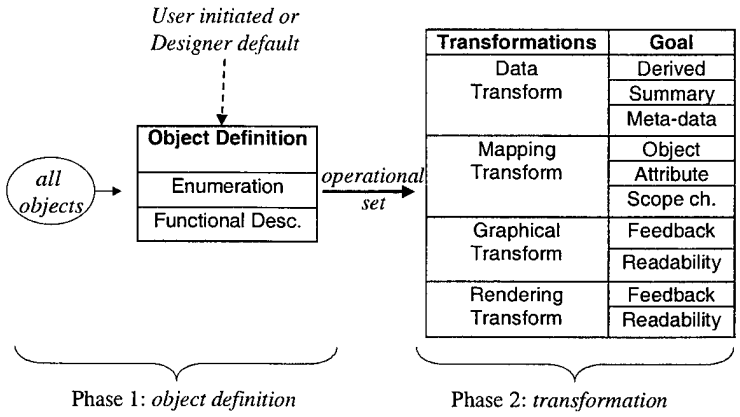


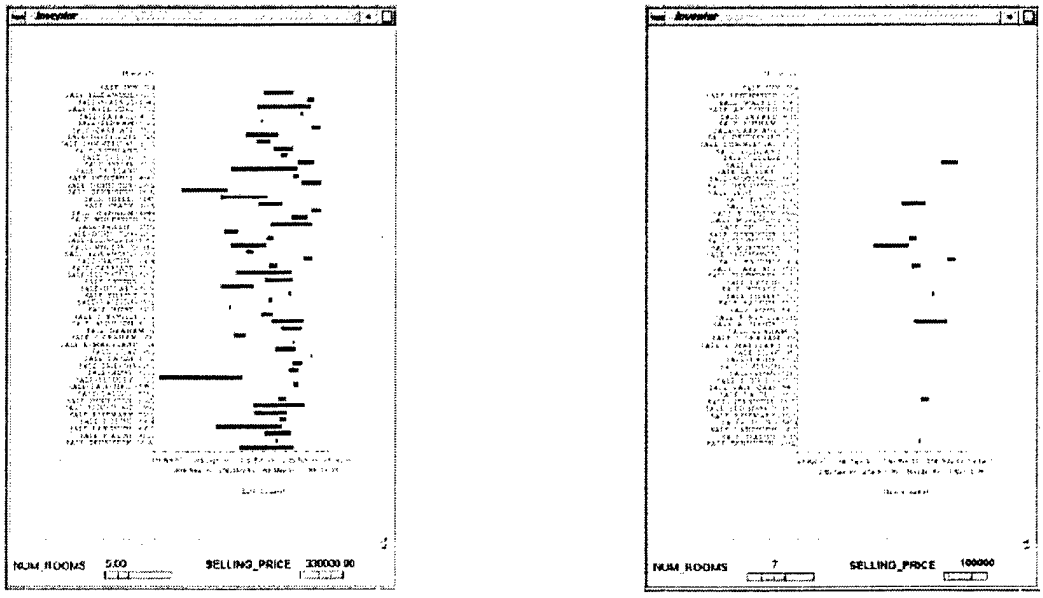
Figure II-3: A visualization technique is defined in this work to contain two components (object definition and transformation). Object definition can be achieved through enumeration or functional description. Transformation can be achieved through data, mapping, graphical, or rendering functions. The transformation functions can be further divided based on their goals.

In this thesis, a *visualization technique* is defined as having two components, an *object definition* component and a *transformation* component (Figure II-3). In the *object definition* component, we define a set of elements that can be from any of the three realms (*data, graphical* or *output media*) in Figure II-1. For example, an interaction may operate over a set of *house* records, a set of graphical *marks* and *bars*, or even a display space within the visualization window. The resulting set of elements from the object definition component (*operational set*) is subsequently processed in the *transformation* component according to our current goals. These transform functions can be used in a multitude of ways to solve different tasks, thus, apart from specifying the transformation class it is

also useful to capture the general goal(s) or effects of these visualization function primitives. The goals of data and mapping transforms, which occur at the start of the visualization generation process, are usually to prepare and set the contents of a visualization. The goals of graphical and rendering transforms, which occur at the end of the visualization generation process, are to provide users with feedback on a visualization technique, or to enhance the readability of the visualization content defined in the data and mapping transform stages.

Visualization techniques allow us to create new visualization designs or modify existing ones. A set of mapping transforms and at least one rendering transform are requisites for creating a new visualization. These transforms are necessary to convert a set of data values to a rendered visual representation of that data. On the other hand data transforms are only necessary if the data set is not in the desired state to begin with and must be further processed. Graphical transforms are needed only when there are readability, feedback or rhetorical requirements. While data and graphical transforms operate within a single realm, mapping and rendering transforms operate across realms, expressing relationships between different object classes. Mapping transforms relate data concepts to graphical objects and rendering transforms relate the graphical objects within a graphical scene to an output media such as the CRT screen. Apart from data, graphical, or media objects, a visualization technique may also be attached to input-devices that allow end-users to interactively alter a technique's functionality or results even if only in a limited way. Techniques that are attached to input-devices are commonly referred to as *interactive techniques*. Subsequently we describe the two visualization technique components: object definition and transformation.

II-1.1 Object Definition Component



(a) Before any data attribute constraints are set

(b) After data attribute constraints are set

Figure II-4: Dynamic Query Sliders applied to house data. Each *bar* encodes a *house data concept*; *x-axis* encodes *date_on_market* and *date_sold*; *y-axis* encodes *house_address*. There are two dynamic query sliders [Ahlberg, 1992], one allows users to place constraints on the *num_rooms* data attribute and the other allows users to place constraints on the *selling_price* data attribute. Houses that do not fulfill constraints become non-visible as in (b).

The object definition component may be initiated by a user or it may be preset as a system default by the designer of the technique. For example, the dynamic query slider technique [*Ahlberg,. 1992*] allow users to search for data elements by placing constraints on their attributes. Constraints are placed by setting data attribute threshold values through the use of input-devices such as *sliders* (e.g. as in Figure II-4). Once the constraints are set, only those data concepts that fulfill the search constraints are shown. Thus the dynamic query slider technique lets users manually initiate object definition by controlling a *slider*.

The visualization system shown in Figure II-5, on the other hand, uses system default objects. Figure II-5 shows a campus map of Carnegie Mellon University with a "Next Lot" button. Pressing this button will cause one campus parking lot to be shown in red. Pressing the button again will cause a different campus parking lot to get highlighted and so on. The particular parking lot to highlight is preset by the system designer. Morewood Parking gets highlighted first, followed by the Parking Garage, etc.

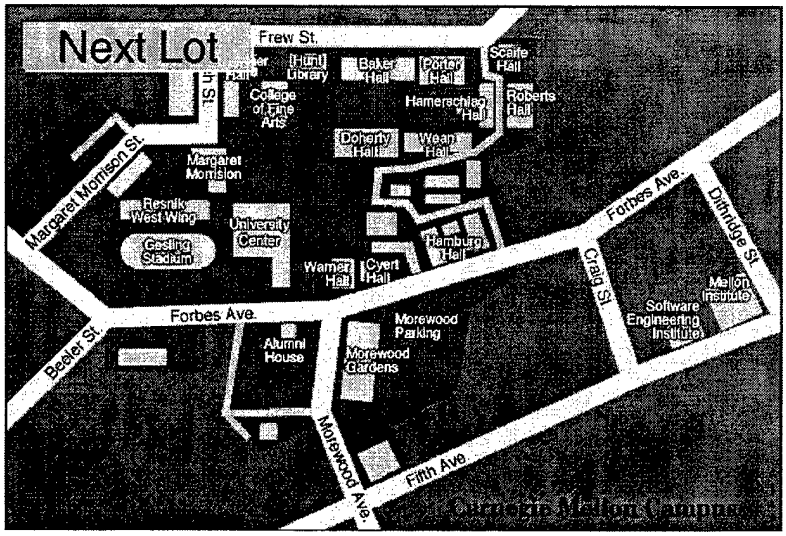


Figure II-5: Map showing the different parking lots at CMU (borrowed from http://www.cmu.edu). Clicking the *next-lot* button will cause a predefined parking lot to get highlighted red (e.g. *morewood parking*). Subsequent presses to the *next-lot* button will cause subsequent parking lots to get highlighted.

There are two primary object definition methods: 1) enumeration or 2) functional description as is shown in Figure II-3. The highlighting technique used in Figure II-5 performs object definition through enumeration because the designer of the graphic explicitly named each and every car park lot on the map. The dynamic query slider technique on the other hand, defines the operational set through functional description. I.e. the object set is captured through a mathematical function applied to object attributes rather than by explicit naming. For this technique, the function used is a simple *greater-than* or *less-than* threshold operator.

II-1.2 Transformation Component

After the operational set has been defined, we manipulate and modify it in the transformation component. There are four classes of transformations, corresponding to the four phases of the visualization generation process: 1) data transforms, 2) mapping transforms, 3) graphical transforms, and 4) rendering transforms. Previous automatic design systems only considered the use of mapping transforms. In this thesis we expand the automatic design process to consider data processing transforms as well. However we leave consideration of graphical and rendering transforms in automatic design for future work. By capturing all four transformation classes in our framework, however, we can more clearly and completely see the roles that data and mapping functions can play in visualization design and in solving data analysis problems.

1. Data Transforms

There are an infinite number of ways with which we can process the information within a data set. As a result a data set may not always contain information that is of interest to us in a form that we desire. By using data visualization techniques we may direct the system to generate derived data attributes that fit our task requirements. Data visualization techniques can commonly be found in spreadsheet programs and data analysis software where a set of different mathematical computations can be applied to selected values in the interface.

2. Mapping Transforms

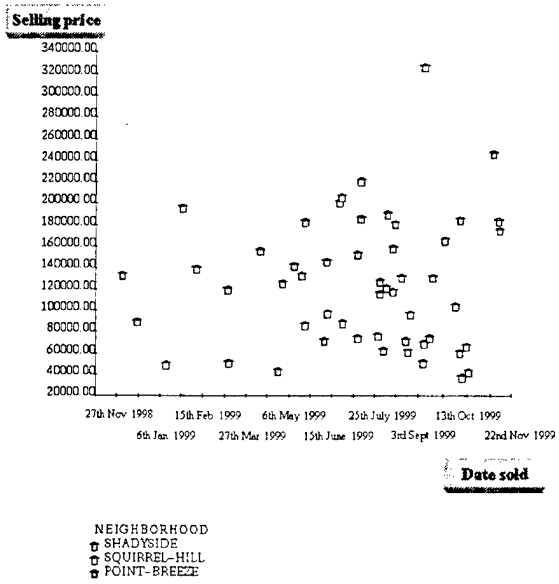


Figure II-6: Visualization system that allows re-mapping of data attributes to the two positional axes. Each *mark* in the visualization represents a *house* data concept. Currently the *x-axis* is set to encode *date_sold* and the *y-axis* is set of encode *selling_price*.

Mapping transforms are most commonly used to encode data concepts and data attributes with graphical objects and graphical properties. To create a new visualization, one or more mapping visualization techniques are

required. Mapping techniques commonly operate on all the data concepts in the data set and the input transformation parameters (e.g. *selling-price, neighborhood, date-sold* and *x-position, color, y-position*) are predefined by the visualization designer. I.e. object definition is achieved through designer default functional description. It is however not necessary for all the mapping transform arguments to be predefined. For example the visualization in Figure II-6 has option menus attached to both the *x-axis* and *y-axis* of the visualization so that users may pick different attributes to re-map to the two axes. Apart from mapping data to graphics, mapping transforms can also be used to change the scope of existing data-to-graphical encodings. For example we could apply the object and attribute mappings in Figure II-6 to only the *pink* objects by using a *scope* mapping transform.

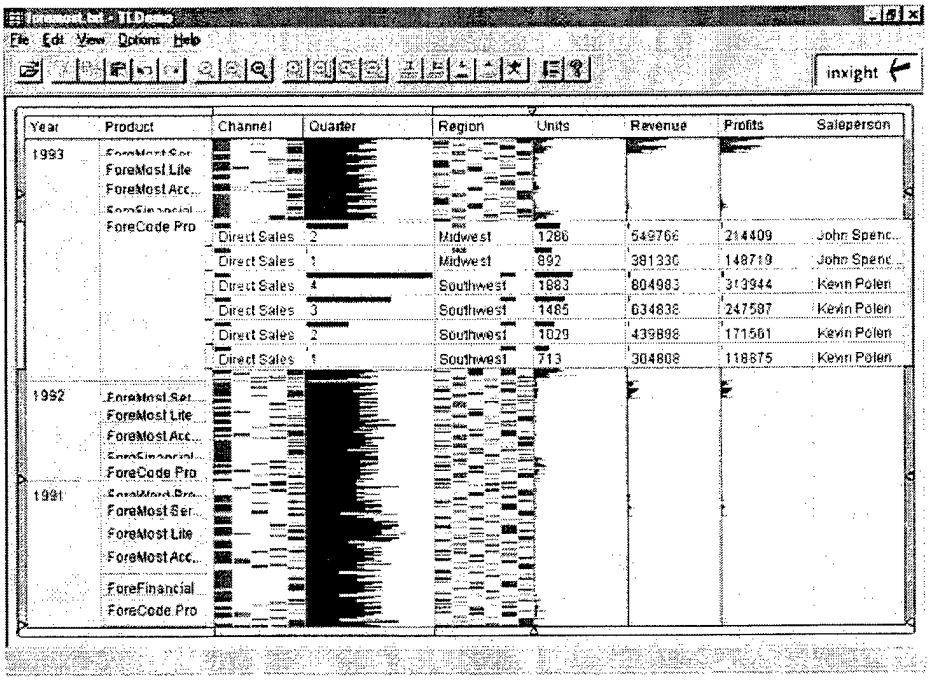


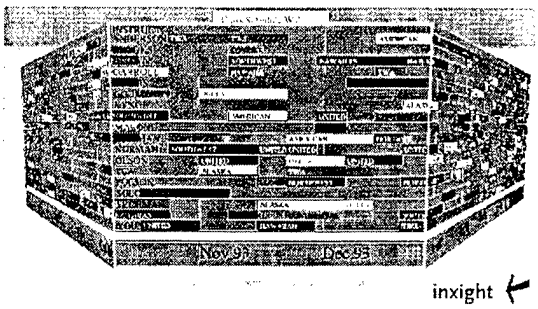
Figure II-7: TableLens System [*Rao, 1994*] (borrowed from www.inxight.com)

Mapping transforms can also be used in more elaborate ways. The PAD++ [*Bederson, 1994*] and the TableLens [*Rao, 1994*] systems use mapping transforms to achieve "semantic zooming". In these systems, the graphical representation classes (e.g. *mark, text, bar*) used in the mapping transforms change based on available screen space. When more screen space is available a more accurate graphical representation is used to show the data and when less space is available a visually simpler but less accurate graphical representation is used instead. For example, the TableLens system (shown in Figure II-7) uses *bars* to represent the data concepts when less space is available and *text* (in addition to *bars*) when the space gets magnified.

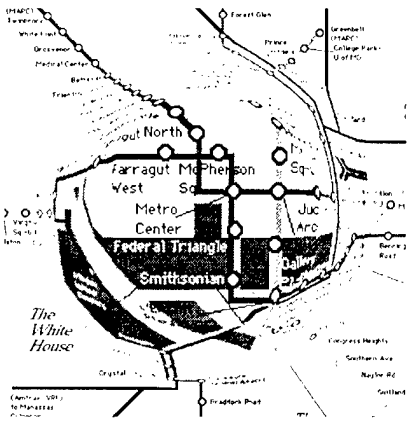
3. Graphical Transforms

Graphical transforms are commonly used to provide feedback or to increase the readability of a visualization. The most common use of graphical transforms is for manipulating un-mapped graphical property values to improve

the readability of a visual design (e.g. layout operations) or for providing simple feedback that reflect state changes (e.g. feedback indicating that a set of objects have been selected). Graphical transforms may also be used as a rhetorical device, e.g. enlarging the objects currently under discussion. Note, however, that graphical transforms are not limited to un-mapped graphical property values. When used to change mapped graphical properties, however, they may distort interpretation of the data set as was discussed in Chuah et al. [*Chuah, 1995*], thus such techniques should only be used with extreme caution and adequate user feedback.



Perspective Wall
(borrowed from Xerox PARC
User Interface Research Group page,
http://www.parc.xerox.com/istl/projects/uir/projects/InformationVisualization.html)



Fisheye non-linear image magnification
(borrowed from
http://www.cs.indiana.edu/~tkeahey/research/fad/fad.html)

Figure II-8: Example rendering visualization techniques

4. Rendering Transforms

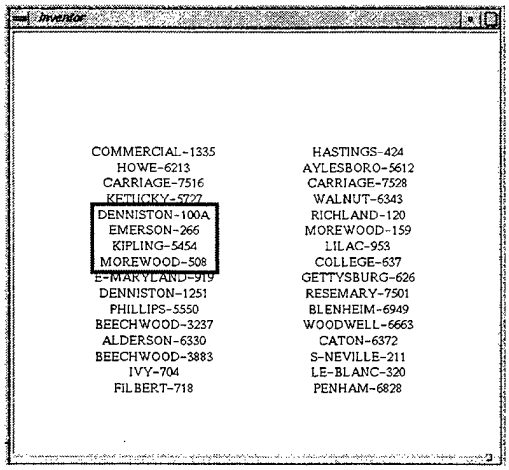
Like graphical techniques, rendering techniques are also used to give users feedback or to improve the readability of a visualization. Many interactive systems today use rendering transforms in interesting ways to distort the graphical scene so that users may focus on particular parts of the scene while maintaining context of the surrounding areas. Some examples include the Perspective Wall [*Mackinlay, 1991*] (shown in Figure II-8a), Fisheye Lens distortion [*Furnas, 1991*] (shown in Figure II-8a), and Table Lenses (shown in Figure II-7). Such techniques improve the readability of visualizations by reducing object occlusion and output density around the focus objects so that their visibility is increased. Apart from distortion techniques, rendering transforms are also used to achieve more common navigation techniques such as *zoom* and *pan*. Appendix F discusses rendering transforms in greater detail and show how they can be used to solve readability problems.

In summary, a primitive visualization technique consists of two components: an object definition component and a transformation component (shown in Figure II-3). The object definition component may be initiated by the user or be set as a system default. There are two ways in which objects may be defined, either through enumeration or functional description. Once defined, the selected visualization elements may be transformed using data, mapping, graphical, or rendering functions. These general transform classes can further be categorized based on their goals or effects.

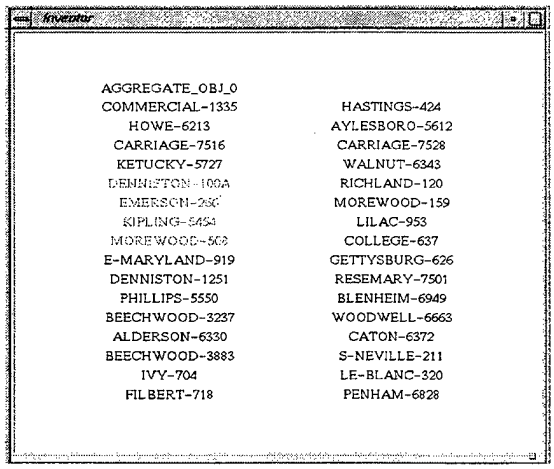
We will build upon this object definition/transformation (*ODT*) model in this chapter and the next. In chapter V we show how we apply this *ODT model* in our prototype automatic design system and how this model allows us to create visualization designs that contain both data and mapping transform functions. The *ODT model*, however, is useful beyond our automatic design system because it enhances our ability to create, and customize visualization techniques, as well as enables us to explore and organize the visualization techniques design space (section II-3).

II-2 Composition

In the previous section we presented a simple ODT model which decomposes a visualization technique into a single object definition operator and a single transformation operator. Many common visualization techniques, however, are more complex and may combine multiple object definition and transformation functions. For example, consider a simple aggregation tool. Aggregation or binning is commonly used for dealing with large data sets. When there are many data concepts, it is difficult to assimilate and analyze all of the data simultaneously. To reduce the data set to a more manageable size, we combine multiple data concepts together and represent them through a single "aggregate" object. There are a wide variety of aggregation methods [*Goldstein, 1994; Chuah, 1998*]. Here we consider a simple aggregation technique that lets users select a set of objects from a visualization display through a *bounding-box*, and then creates a new aggregate object (summary object) containing the selected set. This aggregation process is shown in Figure II-9. First the user selects *denniston-100A, emerson-266, kipling-5454* and *morewood-508* using a *bounding-box* (Figure II-9a). These objects are then grouped to form an aggregate object (*aggregate-object-0*) that appears at the top of Figure II-9b.



(a) Select the set of objects to aggregate



(b) *Aggregate_obj_0*, is generated. *Aggregate_obj_0* contains objects *denniston-100A, emerson-266, kipling-5454* and *morewood-508* that are highlighted in *red*

Figure II-9: Simple aggregation technique. *Text* encodes *house_address*. By using this technique users get to select a set of *house* data concepts using a *bounding-box* and aggregate or group them together to form an aggregate object (e.g. *aggregate_obj_0*). Unlike techniques in the previous section, this aggregation method utilizes multiple transformation methods including a graphical transform to highlight the selected objects and a data transform to summarize the underlying data concepts of the selection.

Object definition through a *bounding-box* falls into the user-enumeration object definition category. Once selected the set of data concepts are summarized using the *group-objects* data transform that maps a set of data concepts to one representative group object as is shown above. Note that in addition to the summarization (*group-data*) data transform, it is useful to give users some feedback as to which objects are being summarized. This is achieved by adding a graphical transform to highlight the selected objects, in addition to the summary data transform. In Figure II-9b, for example, the aggregated objects are highlighted *red*. Such multi-transform techniques are made possible through composition operators that allow us to combine and generate many rich and interesting interactive behaviors from simple two-operator techniques.

There are four main classes of composition: 1) object definition composition, 2) transformation composition 3) producer-consumer composition, and 4) partition composition.

II-2.1 Object Definition Composition

Object definition composition is used when we want to apply the same transformation methods to combinations of multiple object definition sets. Object sets are combined using *set-operator* functions. Set operator functions such as *union*, *difference* and *intersection* take in one or more object sets as input and produce a single output set based on the membership of the input sets.

For example consider the multiple-constraint dynamic query slider technique shown in Figure II-4. This interface has two sliders, one constrains the *selling_price* attribute and the other the *num_rooms* attribute. These two primitive techniques can be expressed using the object definition/transformation model (*ODT model*) as is shown in Figure II-10a. For a complete description of the diagrammatic notations used in these *ODT diagrams* refer to appendix A-1.

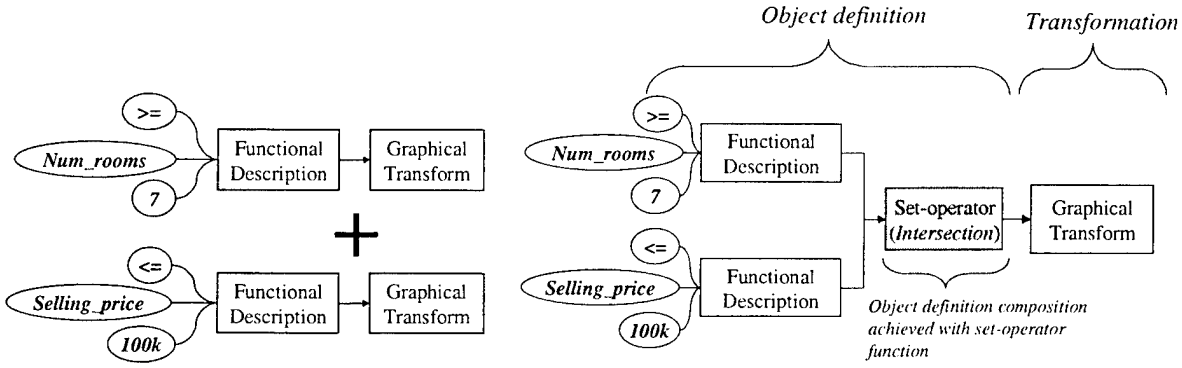


Figure II-10: Object definition composition for the multiple constraint dynamic query technique shown in Figure II-4. The diagrammatic conventions and notations used in the specifications in this chapter and the next are described in appendix A-1.

Expressing the two constraint techniques separately as in Figure II-10a, does not capture the relationship between them that requires both *selling_price* and *num_rooms* constraints to be fulfilled simultaneously. To express this relationship we must perform object definition composition and apply an *intersection* set-operator function to combine the two constraint sets as in Figure II-10b. This will cause only those objects that pass both constraints to be graphically transformed. Set-operator functions are used to further refine selected object sets, thus they are considered part of the object definition component.

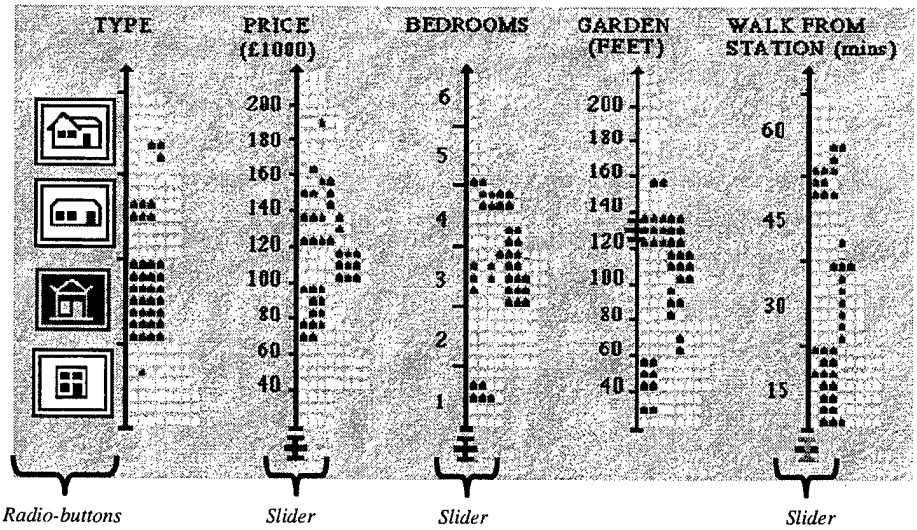
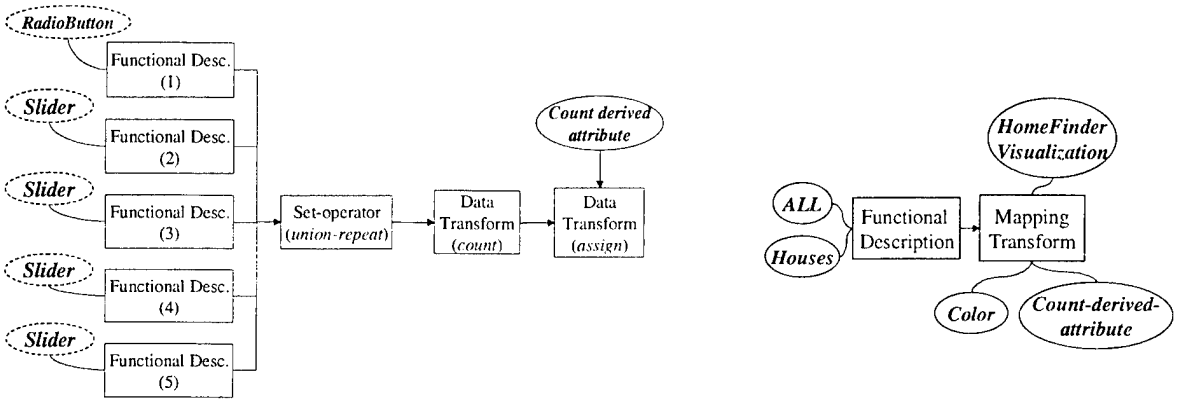


Figure II-11: HomeFinder system [*Tweedie, 1994*] (borrowed from http://infoeng.ee.ic.ac.uk/~lisat/LisaDir/att.html). There are five single-axis aligned charts and a *mark* in each chart represents a *house* data concept. This system uses object definition composition to integrate the object sets selected by the three sliders and the single set of radio buttons. We then *count* the number of times a house appears in the combined set, and use this *count_attribute* to set the color for a given *house* concept.

The type of object definition composition shown in Figure II-10 is quite common. It also occurs in the HomeFinder system [*Tweedie, 1994*] depicted in Figure II-11. In the HomeFinder system each house is represented by a *mark* in each of the charts. There are 5 charts, each with a different house attribute encoded on the *y-axis*. The *x-axis* shows the number of houses that have a particular attribute value. Users get to place constraints on different house attributes using *sliders* and *radio buttons*. The *marks* within each chart are then colored based on the number of constraints passed by the house data concepts they represent. The specification for this method is shown in Figure II-12a. There are several functional description operators in this diagram, each corresponding to a data attribute constraint that is controlled by the *sliders* or *radio buttons*. Each of these constraints defines a set of objects and these sets are combined together using the *union-repeat* set operator. The *union-repeat* operator is similar to the *union* operator except that duplicate objects are not deleted. The combined set is then passed through a *count* data

transform that calculates the number of times a house concept appears in the input set. The results of the *count* function are then associated with each house data concept by using the *assign* data transform operator[2].

Using a separate mapping specification (Figure II-12b), we encode the *count* derived attribute with the *mark color* property in each of the charts within the HomeFinder visualization (Figure II-11). A separate specification is used here because the mapping parameters do not change based on changes in the threshold constraints (i.e. the *count* attribute is always mapped to *color* irrespective of changes in the threshold constraints). Combining the mapping transform with Figure II-12a would create a new mapping each time we change the house selection constraints, and this is not the effect we desire.



(a) In this specification we compute the *count-derived-attribute* which stores the number of times an element appears in the selection set produced by the *union-repeat* operator

(b) In this specification we map the *count-derived-attribute* data attribute, which we computed in Figure II-12a to the *color* graphical property in the *HomeFinder Visualization*.

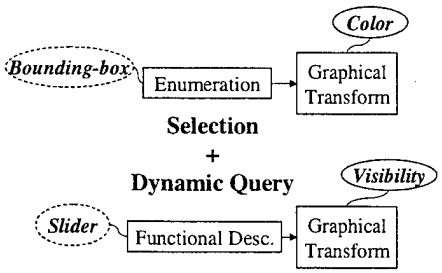
Figure II-12: Functional specification for the HomeFinder system shown in Figure II-11.

We can also apply object definition composition to primitive techniques that have different object definition methods. For example consider the dynamic query slider technique that uses functional description object definition and the object selection method that uses enumeration object description. The *ODT diagram* for both these

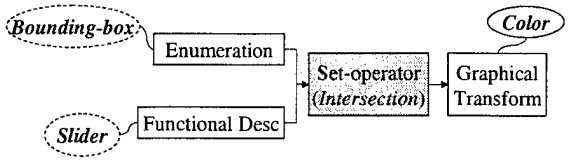
---

[2] We show the *assign* operator here so that we may illustrate how the *count* results change the *color* graphical mappings of the *marks* within the visualization. In general however, we leave out *assign* operators in most of the other technique specifications in this chapter because they do not play an important role in capturing the functional essence of a technique. In the next chapter where we discuss the visualization technique instantiation level, we detail all the instances where *assign* operators are added.

techniques are shown in Figure II-13a. We compose both these techniques using the *intersection* set operator as is shown in Figure II-13b. The technique shown in Figure II-13b only highlights objects that are both enumerated by the user through *bounding-box* selection and that fall within the functional constraints of the dynamic query slider.



(a) Before composition.
The dynamic query technique allow users to search for data elements by placing constraints on a single data attribute by using a *slider* input-device. The selection method allows users to enumerate a set of graphical objects using a *bounding-box*. The selected objects will then be color-highlighted to show that it has been selected.

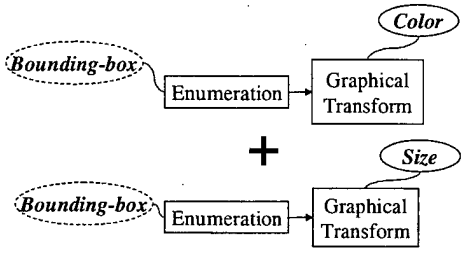


(b) After composition
(the object sets from both techniques are combined using the *intersection* set-operator)

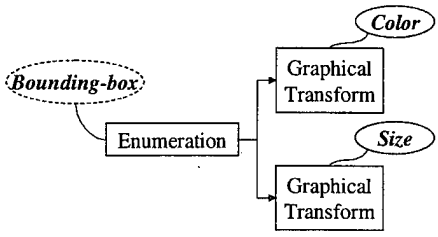
Figure II-13: Object definition composition on the object selection and dynamic query slider technique. The resulting technique ONLY colors those objects that are selected within the *bounding-box* as well as passes the constraint set on the *slider*.

II-2.2 Transformation Composition

Unlike object definition composition, which combines two or more object definition sets and applies the same transformation(s) to the resulting set; transformation composition applies different transformation functions to the same object definition set. For example, consider the object selection technique outlined above (Figure II-13a). Suppose that in addition to highlighting the selected objects *red*, we also want to enlarge them. To achieve this effect we apply two different graphical transforms to the selected object(s) as is shown in Figure II-14b.



(a) Before composition



(b) After composition

Figure II-14: Transformation composition for two different selection techniques with different visual feedback effects. The resulting technique colors and enlarges the objects selected by the *bounding-box*.

We can also perform transformation composition on different classes of transform functions, e.g. a graphical transform and a data transform. For example in the simple aggregation technique described in Figure II-9, we

indicate the objects that have been selected for aggregation by highlighting them, in addition to performing the data grouping. In this technique, transformation composition is applied to the graphical transform for highlighting the objects and the data transform for aggregating the objects as is shown in Figure II-15b.

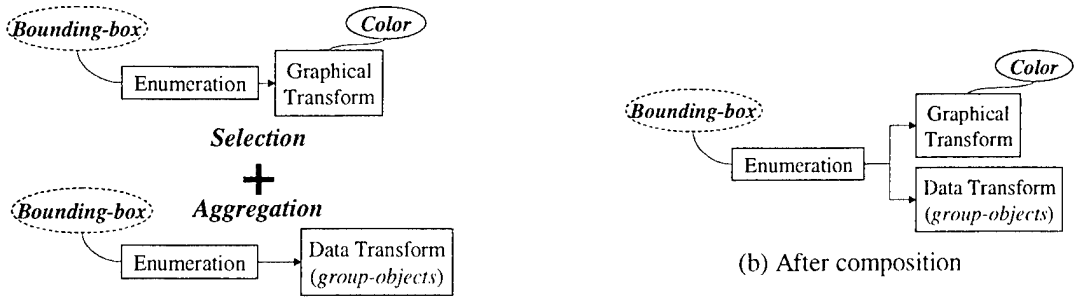


Figure II-15: Transformation composition for the simple aggregation technique shown in Figure II-9. Transformation composition is used here to combine a data transform for creating the aggregate object as well as the graphical transform that highlights the objects within the aggregate, *red*.

II-2.3 Produce-Consumer Composition

Producer-consumer (P-C) composition allows one technique (*producer*) to generate the arguments or information needed by another technique (*consumer*). An example is the *modified value-painting* technique [*Eick, 1992*]. This query method allows users to select a set of objects with an input-device. Chosen data attributes of the selected objects are then summarized and then used to functionally define and highlight another set of objects. The result of such a composition is shown in Figure II-16b where a set of objects is user enumerated through a *bounding-box*. A chosen data attribute (e.g. *selling_price*) of the object set is then summarized using the *mean* data transform. The calculated mean value is then passed to a subsequent functional description operator that selects all objects in the visualization with data attribute values less than the computed mean and then highlights them.

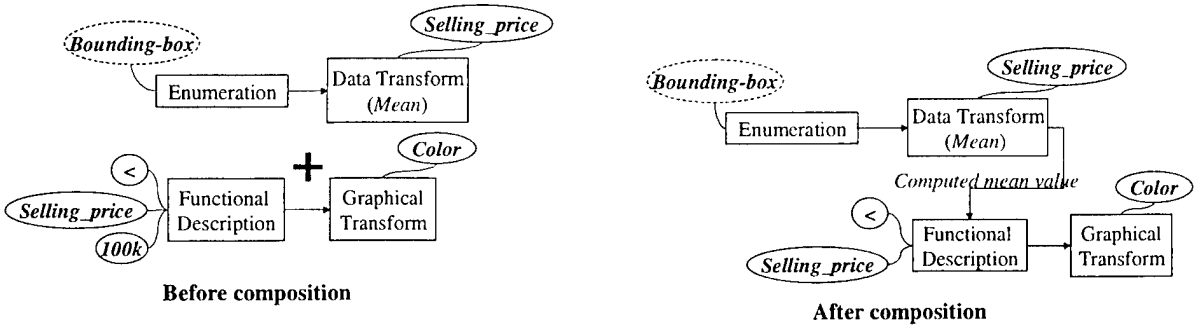
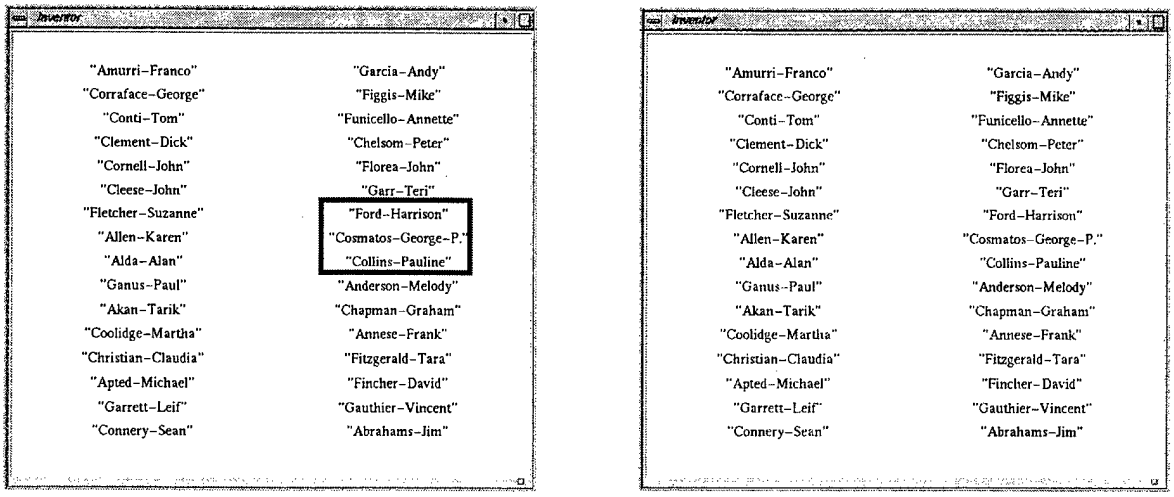


Figure II-16: P-C composition. The computed value from the producer technique (i.e. *mean selling_price*) is piped into the object definition component of the consumer technique and is used to select other objects in the visualization based on the computed *mean selling_price*.

For example, Figure II-17 shows the names of a set of *house_owners*. By using the *modified value-painting* technique, we may select a set of *house_owners* (*Ford-Harrison, Cosmatos-George-P,* and *Collins-Pauline*). Once selected, the technique computes the *mean selling_price* for the houses belonging to the selected *house_owvers.*

Finally the technique highlights all the selected *house_owners* as well as all other *house_owners* with houses costing **less than** the computed *mean selling_price*. Note that unlike the previous two composition classes, PC-composition techniques do not share common object definition sets nor transformation functions.



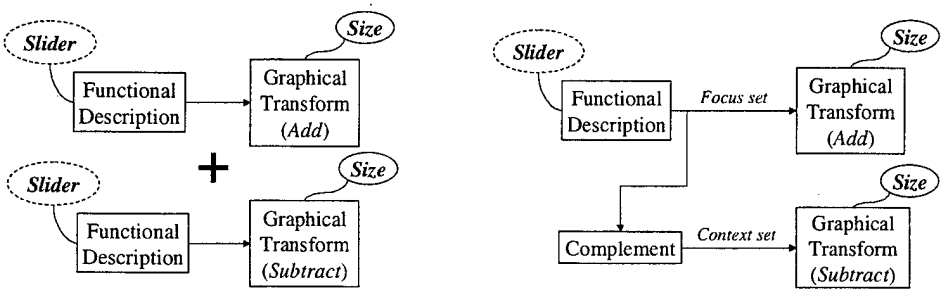
(a) *Bounding-box* used to select set of *house_owners*. In this example, *Ford-Harrison, Cosmatos-George-P,* and *Collins-Pauline* have been selected.

(b) Selected *house_owners* as well as *house_owners* with houses costing less-than the computed *mean selling_price* of the selected *house_owners* get highlighted *red*.

Figure II-17: P-C composition applied to the modified value-painting technique. *Text* here encodes *house_owners*.

II-2.4 Partition Composition

The final class of composition, *partition composition*, is applied when the object definition component of a visualization technique generates more than one set of objects, and we want to transform each generated set differently.



(a) Before composition

(b) After composition

Figure II-18: Partition composition applied to the dynamic query slider technique. Here we use partition composition so that we can enlarge the *focus* objects (i.e. the objects that pass the *slider* constraint) and simultaneously contract objects in the *context-set* (i.e. objects that did not pass the *slider* constraint).

For example we may alter the dynamic query technique slightly so that it enlarges objects that pass the slider constraint and contracts objects that do not. In this case the object definition component generates two object sets,

the *focus set* and the *context set* and each of these partition sets are transformed differently, i.e., the *focus-set* has its size increased and the *context-set* has its size decreased (Figure II-18). Other common ways for partitioning sets are described in Goldstein et al.[*Goldstein, 1994*].

II-2.5 Summary

There are four ways in which primitive techniques may be composed. Each composition type is differentiated by the number and occurrence order of the object definition (OD) and transformation (T) components. Object definition composition, for example, can have multiple object definition components (*n* OD) followed by one transformation component (T); transformation composition has one object definition component (OD) followed by multiple transformation components (*n* T); P-C composition has multiple object definition/transformation components concatenated serially (*n* [OD + T]); and finally partition composition has one object definition component (OD), which generates *n sets* of objects which are subsequently transformed. This information is summarized in Table II-1 below.

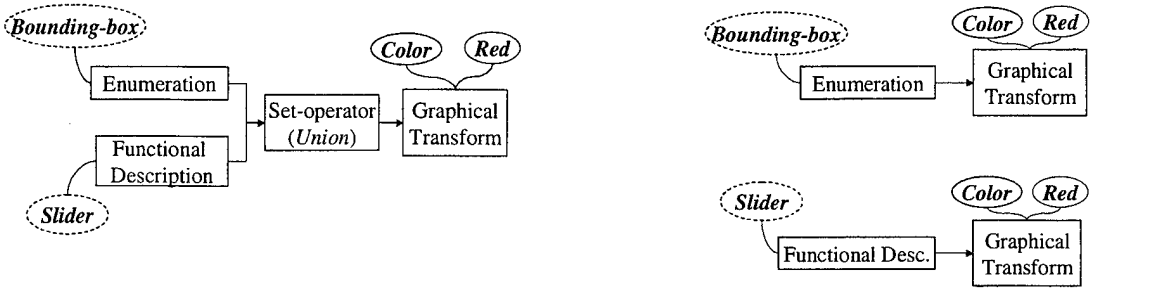
Composition Type	
Object Definition	(*n*OD + T)
Transformation	(OD + *n*T)
Producer-Consumer	*n*(OD + T)
Partition	(OD + *n*[*set* + T])

Table II-1: Summarization of composition types

Even though the four different composition types are described separately in this section, we can apply multiple composition methods simultaneously within the same visualization technique. For example, we may use both object definition composition and transformation composition to apply different object sets to multiple disparate transformation functions. We may subsequently combine the technique through pc-composition and partition composition to additional object definition and transformation functions.

In some cases, the same visualization technique effects may be achieved both with and without composition. For example, suppose we want to highlight objects *red* either by selecting a set of objects through a *bounding-box* or through a dynamic query slider. One way to achieve this is to use object definition composition to union up the selected sets of both techniques and then highlight this combined set *red* (Figure II-19a). Another possibility is to leave both techniques separate as in Figure II-19b. The more appropriate specification depends on our intentions and our conceptualization of the technique. It is usually appropriate to compose two techniques if they are related in some manner, for example, if they are updated by the same input-device event. For example, suppose we want color highlighting to only occur on a *bounding-box* release event, then we would compose the two techniques because they are both triggered by the same input-device and we want to capture this relationship. On the other hand if the

two techniques are not conceptually related, then they should be expressed separately. Composing techniques that are not conceptually related is a misrepresentation of their functionality.



(a) Composed technique. Techniques should be composed when they are conceptually related.

(b) Non-composed technique. Techniques that are not related should have separate specifications.

Figure II-19: Two different technique descriptions that achieve the same effect. Both techniques highlight objects selected by the *bounding-box* or the *slider*, *red*.

This issue of multiple specification solutions extends beyond the simple choice of whether to compose visualization primitives or not. More generally we can sometimes arrive at identical technique functionalities by using different combinations of primitive functions and composition operators. Which design solution is most appropriate would depend on secondary goals such as:

- How does each solution fit with our conceptual model of the technique.
- Which design is more general.
- Which design is more computationally efficient.

Theoretically, compositions allow us to generate an infinite number of designs because we can keep adding more and more operators onto the visualization technique specification. Practically however, the space we are exploring is not infinite. Techniques are only useful when we transform the visualization data and graphical objects in simple and fairly well understood ways. Users have a limited area of attention and a limited amount of cognitive resources. If too much of a visualization is changed, users may miss out on many of those changes; or if the changes are too complex, users may misunderstand or misinterpret the results. In the next section we start exploring the visualization techniques design space using the *ODT model* developed thus far. From this analysis we will see that many common visualization techniques contain very few composition operators and some have none at all. Note that the subsequent section may be skipped if readers are only interested in the use of the *ODT model* with respect to automatic design. Section II-3 explores the generality and coverage of our visualization framework rather than it relates to automatic design.

II-3 Visualization Techniques Design Space

In this section we show the generality and applicability of our framework by using it to evaluate current visualization designs as well as to make meaningful comparisons across techniques in different

visualization systems. Specifically we use our framework to evaluate, compare and classify a set of common visualization techniques, found in Card et al.'s compilation of current visualization systems [*Card, 1999*] and *IEEE's Symposium on Information Visualization* (Table II-2, Table II-4, Table II-6, Table II-8). Some of these visualization systems (e.g. the *TableLens* system) utilize multiple visualization methods, in which case, we represent each of the methods separately in our analysis. Certain other common visualization methods (*dynamic query sliders* and *painting*) appear in multiple visualization systems, in which case, we only describe these base methods once. It is also important to note that while some of the systems analyzed have novel ways of representing and mapping data to graphics, we do not capture these new graphical representation methods here. A representation framework for visualization has been developed previously by Mackinlay and Roth [*Mackinlay, 1986a; Mackinlay, 1986b; Roth 1990*] and our framework defines functional operators that operate on the objects within these previous frameworks. To capture new types of representations we need to focus on expanding the representation framework and that is not the focus of this thesis. Thus only functional techniques and **not** systems with new graphical representations are shown in our analysis tables.

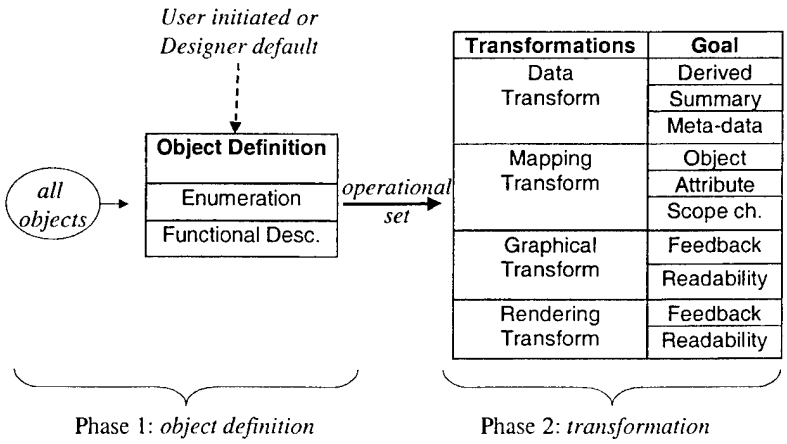


Figure II-20: The two components that form a primitive visualization technique – object definition and transformation

In the following sections we organize visualization techniques according to their main transform classes (i.e. *data transform* techniques are grouped together (Table II-2), as are *mapping* (Table II-4), *graphical* (Table II-6), and *rendering* (Table II-8) techniques). Some techniques may contain multiple transform classes in which case we repeat their specification in each group they belong to. Through this analysis we show that there are some common ways in which visualization techniques are currently used and combined. This knowledge will allow us to better understand visualization techniques and adapt them to more effectively fulfill our tasks and preferences. In addition, we can identify unexplored areas in the visualization techniques design space and start examining new forms of visualization methods. Finally, this analysis also illustrates that our framework is relatively general and is able to characterize a variety of techniques.

In these analysis tables each visualization method is represented by one or more rows. The specific object definition and transformation types of each technique is shown on the table columns. The first two columns represent the two object definition alternatives and the last four columns represent the four transformation classes. A visualization technique is represented by a set of connected highlighted (gray) cells. Highlighting occurs according to a technique's object selection and transformation functions. For example, the dynamic query slider technique has a functional description object definition component followed by a graphical transform that changes the visibility of the objects. As such it has a highlighted cell on the functional description column in Table II-6 that is connected to a highlighted cell in the graphical transform column. The foreground tool, on the other hand, lets users enumerate objects through mouse clicks, thus it has a highlighted cell in the enumeration column of Table II-6. This enumeration cell is connected to a highlighted cell in the graphical transform column, which represents the foreground function that brings selected objects up to the top of a graphical scene. Compositions are represented by double-lined arrows, with a symbol next to it indicating the composition type (OD = object definition composition; T = transformation composition, PC = producer/consumer composition; and PT = partition composition). The dotted arrows in the analysis tables represent *update-links*. *Update-links* indicate changes in a technique that is brought on by changes made in a different technique. For example the HomeFinder system in Table II-2 has an update link from its data transform function to a mapping function. This is because the data transform function updates the *count-derived-attribute* data values that are mapped to graphical objects in the visualization. Thus any change in those *count* values will result in a subsequent update to its graphical value mappings (refer to section II-2 for details). Note that this analysis simplifies the visualization techniques and only includes important functional features, so as to reduce diagrammatic complexity. As a result some of the techniques shown in Table II-2, Table II-4, Table II-6, and Table II-8 may not contain all of the visualization functions from their more complete specifications shown previously in this chapter.

II-3.1 Data Transforms

Data transform techniques are shown in Table II-2. It is perhaps not surprising to note that all these techniques either compose their data transform function with another transform class (*modified-value-painting, generalized-fisheye, simple-aggregation*) or have a subsequent *update effect* (shown as dotted lines) to another transform class (*HomeFinder, TableLens-sort*). Data transforms are commonly connected to a transform function of a different class because data transforms only generate non-visual results, and these results must somehow be shown to users. From Table II-2 we see that data transforms are commonly linked with mapping transforms (*simple-aggregation, TableLens-sort, HomeFinder, generalized-fisheye*). While the *modified-value-painting* technique has a pc-composition link to a graphical transform, there are no techniques with transformation composition links from a data transform to a graphical transform or a rendering transform.

Visualization Technique	Enum.	FD	Data	Map.	Gra.	Ren.
Simple Aggregation (section II-2)			T			
TableLens Sort [*Rao, 1994*]						
HomeFinder System [*Tweedie, 1994*] (section II-2)						
Generalized Fisheye [*Furnas, 1991*]		All	PC			
Modified Value Painting (section II-2)						
			PC			

Table II-2: Data visualization techniques

An example technique that fills the data transform to graphical transform space, is a variation on the selection technique, which we call the *load-sensitive-selector*. This technique allows us to pick a set of data concepts, then highlights the graphical representations of those concepts according to the size (number of concepts) of the selected set. For example, a larger selected set will result in a more saturated highlight color while a smaller selected set will result in a lighter, less saturated highlight color. In this case we have a user-enumerated technique that is linked to a data transform for calculating the number of objects in the set. This count-value is then fed into a color graphical transform for computing the new highlight color. Yet another alternative is to use this count value to change the magnification factor of a rendering transform so that we automatically zoom in when we select a few objects and automatically zoom out when we select many objects. This produces a data transform to rendering transform technique.

Thus by looking at areas not covered by current techniques, we can derive new methods that may be useful for some task classes. The load selection technique for example can be used as a cue to indicate the existence of occluded objects in the selected set. If the saturation is high even though the number of objects that is selected appears to be small then there are probably some occluded objects that have been selected. Note that the new techniques that we discuss in this section (section II-3) are all manually designed based on our analysis of the visualization techniques design space using our framework. In addition these examples tend to be incremental in that they enhance an existing technique in fairly well understood ways. This is so that the expansions can be more easily conveyed and the uses of the expanded techniques are more readily apparent In appendix A-2.2 we present a more complex example of technique expansions that are also based on our framework.

Data transform techniques can be divided into three classes based on the information type generated: 1) derived attributes, 2) summary values, or 3) meta-data. The *generalized-fisheye* technique **computes a**

derived set of *degree-of-interest* (*DOI*) values based on the importance of a data concept to the current task (data transform type-1). Objects with low *DOI* values are subsequently culled from the display with a mapping transform. The *modified-value-painting* and *simple-aggregation* techniques, on the other hand, both **generate summary values** from existing data (data transform type-2). The *modified-value-painting* technique computes the *mean* for a set of values and uses that information through pc-composition for querying other data. The *simple-aggregation* technique summarizes a set of objects into a single aggregate object and then maps the aggregate object into the visualization. Finally the *TableLens-sort* and *HomeFinder* systems both use type-3 data transforms. In *TableLens-sort* the data transform function is used to calculate the **order of elements** (meta-data) based on an existing attribute. The *HomeFinder* system uses a *count* data transform function to generate meta-data for capturing the number of query conditions passed by a set of data concepts.

Note that there are interesting differences between the meta-data generated by the *TableLens-sort* and *HomeFinder* techniques. The former is generating meta-data based on the original data values while the latter is generating meta-data based on a set of query results. A data transform that is attached to a user controlled functional description function (like the *HomeFinder* system) usually falls into the latter category. Query data transform techniques are commonly transient, because their purpose is to give users one-time feedback on the current query results. As the query changes, new query information is computed/summarized and the previous results are discarded. In addition, because we are using data transforms to get an overview of a user initiated query, we only use *summary* and *meta-data* transform functions (e.g. to calculate the size of query set, the number of duplicate elements within the query set, the spatial dispersion of objects within the query set, etc). This separation between *query* and *non-query* data transforms is shown in Table II-3.

			Visualization Techniques
Query (transient) *(data transform connected to a user defined functional description object definition method)*		Summary	
		Meta-data	*HomeFinder*
Non-Query	Final results (persistent)	Derived	
		Summary	*Simple-aggregation*
		Meta-data	*TableLens-sort*
	Intermediate results (transient)	Derived	*Generalized-fisheye*
		Summary	*Modified-value-painting*
		Meta-data	

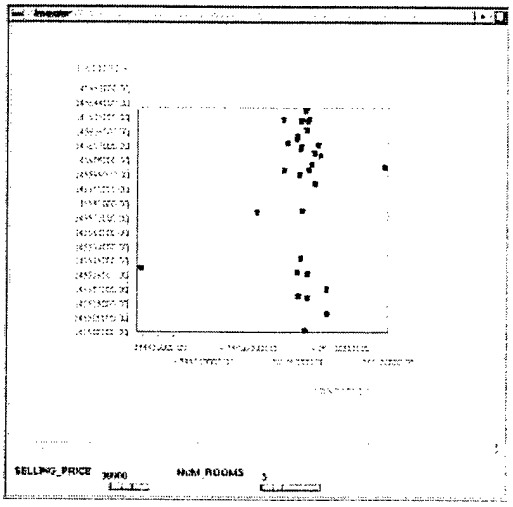
Table II-3: Goal categories of data visualization techniques

Non-query related data transforms are divided into two groups: a) those that compute final results, and b) those that compute intermediate results. Intermediate result techniques use data transforms to calculate

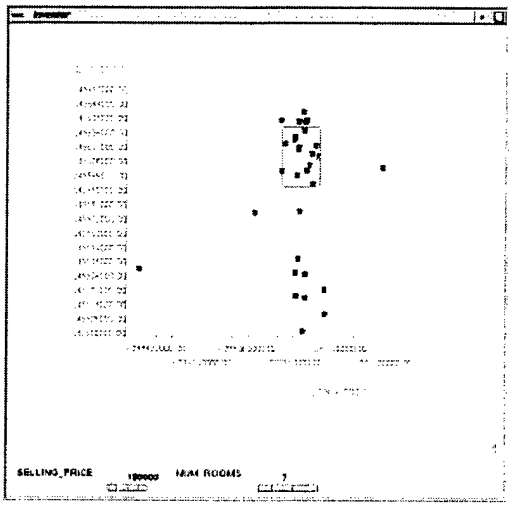
temporary values that are used in subsequent functions. Thus, intermediate-result techniques are always linked either through pc-composition or transformation composition to other functions. An example is the *modified-value-painting* technique that uses a *summary* data transform to get the *mean* of a value set. This mean value is subsequently used in a functional description function. Another example is the *generalized-fisheye* technique that computes a set of intermediate *DOI* values, which are subsequently filtered with a functional description operator. Note that intermediate operations are also commonly used in complex computations. For example, to get the result for (A – B) + (C – D), we must first calculate (A – B) and (C – D), which are both intermediate results. These results are then fed into an addition data transform that produces the final result.

Final-result techniques, on the other hand, have more persistent output values that are commonly reused multiple times in several different tasks. In this case we are extending our database with new information. In the other two cases (i.e. query and non-query-intermediate) we are not looking to extend our database but rather just to summarize feedback results in a way that is easily assimilated by users or to fulfill intermediate tasks. Table II-3 show how the data transform techniques in Table II-2 may be classified based on these different classes of data transform goals.

In Table II-3 we see that none of the techniques considered fall into the query-summary category. One example technique we manually designed that fills this space is the *range-dynamic-query* technique that presents end-users with the spatial range of objects that fall within the query set.



(a) The *red bounding-box* encapsulates all the objects within the visualization because no constraints are currently set.



(b) The *red bounding-box* now only encapsulates objects that are on the top-mid portion of the display because only objects in that area pass the two *slider* constraints.

Figure II-21: Range dynamic query technique. In the example visualizations above, each *mark* represents a *house* data concept. The *x* and *y positions* of the *marks* corresponds to the geographic location of their respective *houses*. Objects are selected here by setting constraints using the two sliders at the bottom of the interface, which allow users to set threshold constraints on the *selling_price* data attribute and the *num_rooms* data attribute. The red *bounding-box* is then drawn so that it encapsulates all of the selected objects (i.e. all objects that pass the *slider* constraints).

For example Figure II-21 shows a *range-dynamic-query* interface that allows users to select objects based on *selling_price* and *num_rooms* using two *slider* input-devices. The visualization encodes geographic location of the houses on its *x-axis* and *y-axis*. In this interface we show summary information for the query objects by drawing an *annotation-box* (*red* rectangle) around all houses that fall within the query set. This is achieved by using the *get-max* and *get-min* summary data transforms to get the minimum and maximum *longitude* and *latitude* of the query objects. These data transform functions are later connected to a graphical transform function for changing the size of the *annotation-box*. From Figure II-21 we see that there is a relationship between house *selling_price* and *num_rooms* with *location*. The more expensive larger houses seem to be located in the top middle portion of the area. This technique is useful in cases where the visual display is large and it is difficult to focus in on the selected objects. By using the range dynamic query slider, we can automatically zoom in on the display based on the range of the selected set. This would work well in those cases where the query result is highly correlated with object positions in the visual display.

In Table II-3, the non-query-final-derived and non-query-intermediate-meta-data categories are also empty. However, all non-query data transform techniques (including the two empty ones in Table II-3), can commonly be performed in spreadsheet and data analysis systems which allow users to compute a wide range of non-query results based on user input formulaic expressions.

II-3.2 Mapping Transforms

	Visualization Technique
Pure mapping techniques	Visage Drag and Drop [*Roth, 1996*]
	Dipstick [*Beshers, 1990*]
Rendering -mapping techniques (Semantic zooming)	PAD++ [*Bederson, 1994*]
	TableLens Distortion [*Rao, 1994*]
Data-mapping techniques	Simple aggregation (section II-2)
	Generalized Fisheye [*Furnas, 1991*]

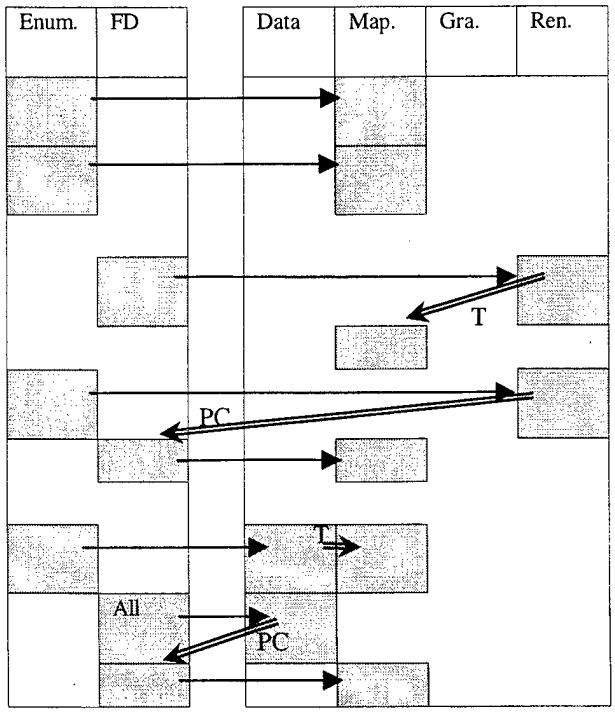


Table II-4: Mapping visualization techniques

The mapping techniques we analyzed are shown in Table II-4. Mapping transforms, as was discussed previously, are commonly associated with data transforms (*generalized-fisheye, aggregation, etc*). While there are several *semantic zooming* techniques (*PAD++* and *TableLens-distortion*) that compose mapping transforms with rendering transforms, there are no techniques in Table II-4 that combine mapping transforms and graphical transforms. An example technique we designed that fills this space is the *information-enhancer* technique that allows users to select a set of focus data concepts, and then increases the number of data attributes mapped (i.e. the amount of information encoded) for those chosen concepts (mapping transform) (e.g. similar to semantic zooming techniques). At the same time however, we also increase the size of those objects (graphical transform) so that the additional information encoded can be viewed more clearly and accurately. The *TableLens-distortion* and *simple-aggregation* techniques in Table II-4 can also be expanded with a graphical transform for *coloring* the operational set (i.e. the objects being transformed) so that they appear more salient to users.

Mapping transforms are used first and foremost to show end-users the information required for solving their task(s). All visualizations use mapping transforms for this purpose. Some visualization techniques also use mapping transforms to change the content of visualizations in order to improve readability[3]. The contents of a visualization can be set or changed by 1) mapping data concepts to graphical objects, 2) mapping data attributes to graphical properties, or 3) changing the scope of existing mapping functions. Table II-5 shows how the mapping transform techniques shown in Table II-4 can be classified based on these goals.

		Visualization Techniques
Show task data	Object	*All visualization systems*
	Attribute	*All visualization systems*
	Scope	*Visage-drag-&-drop*
Readability	Object (*semantic zooming*)	*TableLens-distortion, PAD++*
	Attribute (*semantic zooming*)	*TableLens-distortion*
	Scope	*Visage-drag-&-drop, Dipstick, Generalized-fisheye, Simple-aggregation*

Table II-5: Goal categories of mapping visualization techniques

[3] Readability refers to problems that arise due to constraints of the visualization output media and constraints of our perceptual system. These problems reduce the effectiveness of a given visualization design because of object occlusion, ink density, lack of information presence, and dwarfed encoding scales.

Visualization *readability* can be improved through *semantic zooming* techniques as well as *scope* techniques. Semantic zooming techniques improve the readability of a graphic by allowing users to view different pieces of information at different degrees of detail. This is achieved by using different object and attribute mappings at different instances. When data sets are large, it is not possible to show all the information in detail without overwhelming the user and overcrowding the available display space. One way to solve this problem is to represent the information with simple graphical objects that take up little space and only map the current focus information to richer graphical objects. Usually a change in graphical object representation is coupled with one or more attribute mapping changes as well. For example in the *TableLens-distortion* technique, we change the graphical representation used from *bar* to *text* (and *bar*) when the space available for showing the data is expanded. We also use the *label* graphical property in addition to *bar-lengths* for showing the data values.

Scope techniques, on the other hand, use mapping transforms to change the data concept or graphical object set being shown in the visualization but not the graphical representation class or graphical properties used, as was done in semantic zooming. The *visage drag and drop* technique [*Roth, 1996*] is a scope technique that allows users to select and transfer sets of objects from one visualization to another by changing the set of data concepts being encoded within each visualization. Operations available in the "*Data source*" menu of a *Microsoft Excel* chart uses mapping transforms in much the same way. To add objects into a chart users select a set of data concepts from the spreadsheet and then choose the "*Add*" option in the "*Data source*" menu. Both of these scope changing techniques can be used to improve visualization readability only showing the relevant data concepts at any given instance, thereby reducing clutter and ink density. However, using mapping transforms in this way to solve readability goals requires that we keep track of how we changed the contents of the visualization so that we can restore deleted information that may be needed at some later point of our task. In contrast, the *generalized-fisheye-lens* technique automatically changes the scope of visualizations based on a *degree-of-interest (DOI)* function and thus does not require users to manually control the object scope, but automatically adds and deletes objects as necessary. As a result, unlike the *visage-drag-and-drop* technique, we do not have to keep track of scope changes made to the visualization. However, it can be difficult to devise appropriate *DOI* functions for our task(s). Scope techniques are also often combined with data summarization operators (e.g. in certain aggregate methods) to add in new summary information and delete the original objects or values that have been summarized. This allows us to improve visualization readability because the number of objects shown are reduced.

II-3.3 Graphical Transforms

Graphical transforms (shown in Table II-6) are used to provide feedback or improve the readability of visualizations. In Table II-6, we see that graphical techniques are not commonly combined with other transformation classes. In the previous sections we had discussed several possible new design alternatives for combining data and graphical transforms as well as mapping and graphical transforms. Here we discuss designs that combine graphical and rendering transforms. Techniques do not usually combine graphical and rendering transforms because both of these function classes serve similar purposes, i.e. to provide feedback or to improve readability. In addition, rendering transforms may sometimes warp the effects of a graphical transform that operates on the same graphical

property class (e.g. *positional, retinal*) making the resulting visual effect of the combined technique difficult to interpret. Thus a design that attempted to combine both graphical and rendering functions through transformation composition would do well to apply the graphical transform to different graphical property classes. For example, we could augment the *TableLens-distortion* technique (which operates on *positional* graphical properties) with a graphical transform that colors all objects within the lens *red* (i.e. operates on *retinal* graphical properties). This highlights the selected cells so that users can more easily tell which objects are being expanded. The resulting specification has mapping, graphical, and rendering functions combined together through transformation composition. Another alternative is to combine graphical and rendering functions using pc-composition. For example, to reduce occlusion, we could render all graphical objects that exceed a threshold *area* as wire-frame objects instead of solids. This technique uses a graphical transform to compute the *areas* for all graphical objects within the visualization. The *area* values are then piped into a functional description object definition function that ultimately leads to a wire-frame rendering transform.

	Visualization Technique
Feedback	Dynamic Query [*Ahlberg, 1992*]
	Value Painting [*Eick, 1992*]
	Modified Value Painting (section II-2)
	Simple selection (section II-2)
	Painting [*Becker, 1987*]
Readability	Foreground Tool
	SDM general [*Chuah, 1994*]
	SDM distance [*Chuah, 1994*]
	Shrimp [*Storey, 1997*]
	TableLens Column Move [*Rao, 1994*]
	Cone Trees [*Robertson, 1991*]

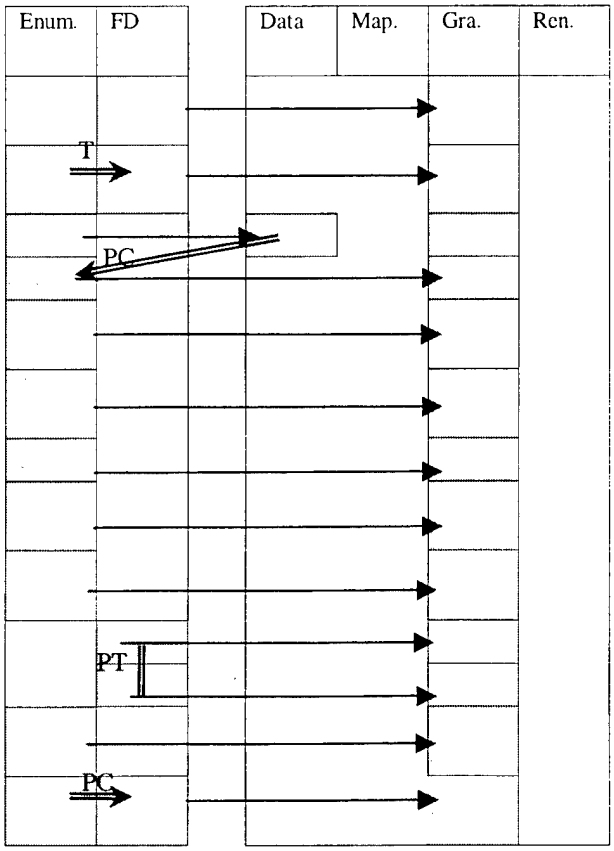


Table II-6: Graphical visualization techniques

There are two types of graphical technique feedback as is shown in Table II-7: search feedback and internal state feedback. Search-feedback techniques use graphical transforms to make the results of a query more salient so that a user's attention is drawn to those objects. Some example search techniques include *dynamic-query-sliders,*

and *value-painting*. All of these techniques allow users to specify a functional description of the data concepts they are interested in. The appearance of the graphical objects representing the specified data concepts are then colored, made visible, enlarged, etc, to give feedback to users on the results of their search. The graphical transforms used for search and internal-state feedback are often simple such as setting a group of graphical values to a constant. This is however not a requirement. In appendix A-2.2 we consider a hybrid search technique with complex feedback effects involving many graphical transform operators.

		Visualization Techniques
Feedback (*commonly simple feedback)*	Search feedback (*functional description object definition*)	*Dynamic-query-sliders, Value-painting*
	Internal state updates (*stored internal information that is not readily apparent*)	*Object-selection, Simple-painting*
Readability	Occlusion	*Foregound-tool, SDM-distance, Shrimp*
	Ink density	*Shrimp*
	Information Presence	*TableLens-column-move, SDM-distance*
	Dwarfed Encoding Scales	*SDM-size-general*

Table II-7: Goal categories of graphical visualization techniques

Internal-state-feedback techniques use graphical transforms to encode internal state information and relationships among objects that may not be apparent in the visualization. *Object-selection* and *simple-painting* are some example internal-state-feedback techniques. For object selection, feedback is required to give users persistent state information on the object set that he/she is currently controlling. This is especially important when we have multiple object sets, or when we create an object set by slowly adding objects into it (many applications allow users to do this by using *shift-click*). In both these cases it is difficult to keep track of the objects that are currently selected, thus visual feedback is required to show users the current "*selection state*". The *simple-painting* technique, on the other hand, uses graphical transforms to reveal object relationship state-information. Specifically, graphical feedback is used to show the mapping relationships between certain data concepts and graphical objects (i.e. which graphical objects correspond to a given set of data concepts).

Readability graphical transform techniques change the effectiveness with which users can view the graphical elements that are already contained within a visualization. This is in contrast to mapping techniques that change visualization readability by altering visualization content. We focus on four primary readability problems: 1) occlusion, 2) ink density, 3) information presence, and 4) dwarfed encoding scales. These readability issues are

described in greater detail in appendix F. Unlike feedback techniques, readability enhancing techniques such as *Shrimp, foreground-tool, SDM-distance-operation,* and *TableLens-column-move* tend to have an enumeration object definition component. This is because readability problems are often the result of complex spatial relationships among the objects and it is difficult to capture the set of objects involved, functionally. There are several different ways in which readability may be improved. The *foreground-tool* improves visibility by reducing occlusion for a set of interesting objects; the *Shrimp* system enhances the visibility of focus objects by making them larger in size, and reducing the density of elements around the focus objects (this may sometimes remove occlusion problems as well). The *SDM-distance-operation* and *TableLens-column-move* techniques address the information presence readability issue. The *SDM-distance-operation* allows users to bring a set of objects closer together and facilitate comparisons by transforming the objects spatially so that they fall on a straight vertical line perpendicular to the user's point of view. The *TableLens-column-move* technique, on the other hand, allows users to move table columns closer together so that the values are more easily comparable. Note that the *SDM-distance* technique also has the side effect of removing occlusion from the focus set, as well as enlarging the focus objects by bringing them closer to the user.

II-3.4 Rendering Transforms

Finally, like graphical techniques, rendering techniques (shown in Table II-8) are also used to provide feedback and improve visualization readability.

Visualization Technique
Zoom
Pan
Graphical Fisheye [*Furnas, 1991*]
Stretching [*Sarkar, 1993*]
Bifocal Lens [*Leung, 1989, 1994*]
TableLens Distortion [*Rao, 1994*]
PAD++ (rendering effect is similar to zoom) [*Bederson, 1994*]

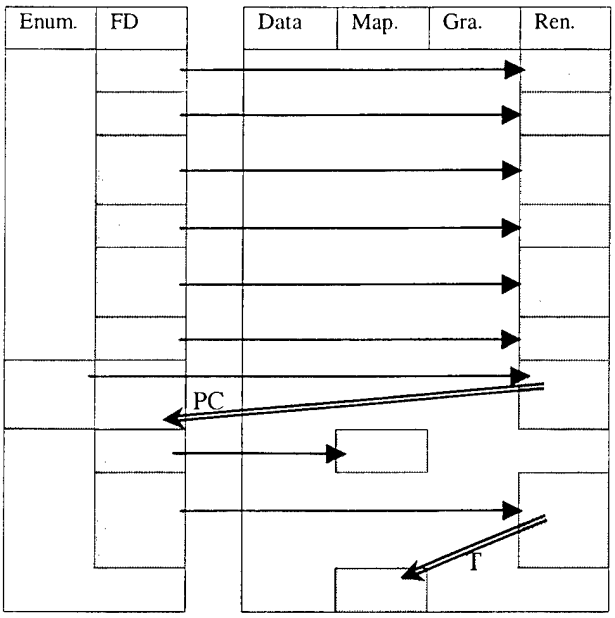


Table II-8: Rendering visualization techniques

Rendering techniques transform the way with which a visualization graphical scene is drawn on the CRT display, so that users may focus their attention on different parts of the scene that are pertinent to their current task. Because of the nature of such transforms, most rendering functions operate on the entire visualization window

[*Furnas, 1991; Leung, 1989,1994*]; transforming only sub-regions will cause discontinuities in the display space that are often distracting, and result in a loss of information [*Hollands, 1989*]. Example techniques that do not operate on the entire visualization window includes the *lens distortion* techniques [*Bier, 1994; Rao, 1994*].

As can be seen in Table II-9, rendering techniques are more commonly used to solve readability issues rather than feedback. One reason is that feedback techniques usually only require simple visual stimuli whereas rendering transforms tend to bring about more complex visual changes. There are two main classes of rendering techniques: *navigation techniques* and *distortion* or "*focus + context*" techniques. Navigation techniques such as *zoom, pan* and *scrolling* allow users to view different areas of the information space when the output media is not sufficiently large to contain all the information simultaneously (i.e. information presence readability issue). Note that as is shown in Table II-9, *zoom* techniques can be used to solve other readability problems as well. The main problem with *navigation* techniques, however, is that users may easily lose track of where they are in the information space.

			Visualization Techniques
Feedback			None analyzed
Readability	Navigation	Occlusion (only *perspective occlusion*)	*zoom, pan*
		Ink density	*zoom*
		Information Presence (view different information)	*zoom, pan, scrolling*
		Dwarfed Encoding Scales	*zoom*
	Focus+context	Occlusion (only *perspective occlusion*)	*graphical-fisheye-lens, bifocal-lens, stretching*
		Ink density	*graphical-fisheye-lens, bifocal-lens, stretching*
		Information Presence (more information to be shown)	*graphical-fisheye-lens, TableLens-distortion, bifocal-lens, stretching*
		Dwarfed Encoding Scales	*bifocal-lens, TableLens-distortion*

Table II-9: Goal categories of rendering visualization techniques

Focus + context techniques or spatial distortion techniques, such as the *graphical-fisheye-lens, bifocal-lens, stretching* and *TableLens-distortion* techniques warp the visualization space so that less output space is given to the information on the periphery and more output space is given to the focus objects. By reducing the contextual areas, these techniques can be used to show more information at any one time compared to non-distorted displays (i.e. greater information presence) and as a result it is less likely that users will get lost in the information space. In

addition, the expanded focus area(s) help reduce ink density around focus objects so that they can be viewed more clearly. In 3D-displays, these distortion techniques may also be used to remove *perspective-occlusion*. Note that even though *TableLens-distortion* is a focus+context technique, it does not appear in the occlusion and ink-density categories of Table II-9 because these two readability problems are not applicable to the *table* representation used in the *TableLens* system. Finally, focus+context techniques may also be used to expand dwarfed encoding scales (i.e. a positional axis that is too small in size to accurately represent all of the data values). However, their non-linear magnification functions distort the encoding scales, making them difficult to interpret. The two exceptions are the *TableLens* and *Bifocal-lens* techniques, which use very simple distortion functions. In these techniques the display only has two different distortion scales, one for the *focus area* and one for the *context area*; unlike the other techniques which apply a continuous distortion function. As a result any distorted axes will also only have a focus and a context section, and as such are easier to interpret.

II-3.5 Summary

In this section we have classified a set of common visualization techniques using the object-definition/transformation structure described in previous sections. Based on this classification we were able to draw some interesting similarities among the different transformation techniques and categorize them based on their goals as is shown in Table II-3, Table II-5, Table II-7, and Table II-9. We were also able to recognize the common ways in which current visualization techniques are used, how they are commonly composed with other functions, and the areas in the design space that have yet to be rigorously explored. In fact, in this section we presented some simple adaptations of current techniques (e.g. *load-sensitive-selector, range-dynamic-query, information-enhancer*) that fall within some of the less populated design areas. These techniques are interesting, if not in their end-use, then in filling out the visualization design space and in illustrating the strengths and weakness of new technique classes that have never been explored. Note that in this section we do not discuss how "good" a technique is at fulfilling its intended task. The goodness of a technique with respect to a task is evaluated in chapter IV-2, using four different measures: articulatory, functional, expressive, and observational distances. These four distances determine which visualization techniques are most appropriate for a task by estimating the amount of motoric, cognitive, and perceptual effort users must expend to solve their tasks.

II-4 Conclusion

In this section we decomposed visualization techniques into two components: an object definition component and a transformation component. In the object definition component users pick a subset of data and graphical elements from all the available elements in the visualization system. In the transformation component, different transform functions are applied to the objects from the object definition component in order to bring about different visualization effects. These simple two-component primitive visualization techniques may be composed with each other using four different classes of compositions: object definition composition, transformation composition, producer-consumer composition, and partition composition. Each of these composition types are characterized by

different patterns of object definition and transformation chains. Based on the *ODT model* (object definition/transformation) and composition functions, we get a better idea of what constitutes a visualization technique, how they are built, combined and used. This is crucial to our automatic design system and we show how the concepts set forth in this chapter are applied to our prototype design system in chapter V. In addition, we can also use our framework to scope out the design space for common interactive techniques (Table II-2, Table II-4, Table II-6, and Table II-8) and make comparisons across techniques from different visualization systems. These tables show the flexibility of our framework in being able to represent a wide range of current visualization techniques. It also reveals the ways in which common techniques are used and combined so that we are better aware of current design boundaries and where the unexplored areas are within the visualization techniques design space.

This chapter focussed on examining visualization techniques at a higher, functional level of abstraction where we are mostly concerned with the general class of object definition and transformation functions used, as well as their goals. There is little discussion here of inputs and outputs to the functions, how these functions get their needed input arguments, and how input-devices can be integrated into the design. These issues must be solved before we can use our visualization techniques framework in our automatic design system to render or generate a working visualization interface. We consider these instantiation issues in the next chapter.

Chapter III:
Visualization Techniques Framework

A
Concrete Instantiable Description

In the previous chapter we discussed visualization techniques at an abstract, functional level. This level of specification is useful because it captures the general purpose and capabilities of a technique. The functional specification also allows comparisons of techniques without the added complexity of interface and structural details. This level of specification however is insufficient to capture a technique in enough detail for instantiation (i.e. rendering it as a visualization interface). In order to increase the applicability and practicality of our framework, we must also describe how the functional specifications described in the previous chapter can be augmented to an instantiable form.

Other previous frameworks on visualization techniques either focussed on the functional level (e.g. [*Tweedie, 1997*], [*Card et al., 1997*], [*Card et al., 1999*]) or the instantiation level (e.g. Data explorer, IRIS Explorer, AVS [*Brodlie, 1991*]) in isolation. In this thesis it was crucial for us to describe a framework that encapsulates both levels of description. The functional level informs our automatic design system of the primary primitives within a visualization technique, their uses, and how these primitives may be combined with each other. The instantiation level allows our designer to describe the structural details of a technique (e.g. data and graphical elements, input-devices, etc) so that the technique can be rendered as an active interface. By including both levels in the same framework as well as a method of transition from one level to the other we provide the advantages of both types of description without requiring a change in the conceptual model or descriptive language as we would with previous frameworks. In addition this two level design methodology is also useful to visualization designers because it encourages them to initially focus on the functional aspects of visualization techniques, free from structural constraints. Once the functionality has been fully designed, a designer may enrich the technique with structural detail. As a result designers will be less apt to falsely constrain functional capabilities as a result of structural concerns. For a more complete discussion of how our framework differs from previous work refer to appendix B-1.

In this chapter we present a five step process, which we call the *instantiation augmentation process*, that can convert any functional specification into an instantiable visualization technique (Figure III-1[1]). A *functional* specification (like the ones shown in the previous chapter) captures the "core" functionality[2] of a technique in as general and abstract terms as possible. An *instantiation* specification augments a *functional* specification with specific input-devices and translation functions[3] that are required to make the technique operational. The five steps in the instantiation augmentation process dictate the general look or structure of the technique but does not change its underlying functionality. For example, the topmost diagram in Figure III-1 shows a functional specification for the object highlight technique, which graphically transforms a set of user selected objects. The functional specification is very simple consisting of an enumeration selection method followed by a graphical transform. The instantiation specification of this technique (shown at the bottom of Figure III-1) is more complex, consisting of various input-devices (e.g. *bounding-box*) and translation functions (e.g. *get-values* function) in addition to the object definition and transformation functions used in the functional space.

Next, we describe the five steps of the *instantiation augmentation process* shown in Figure III-1. The gray highlights in Figure III-1 indicate changes made to the specification at each step.

Step1. *Pick specific object selection and transformation functions*:

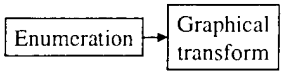
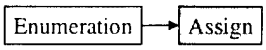
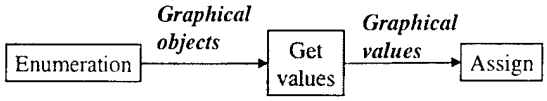
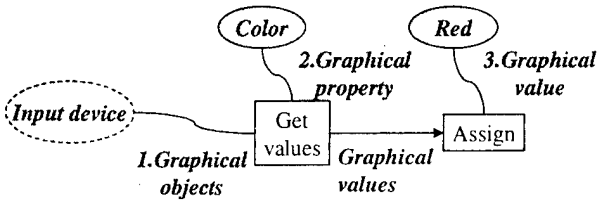
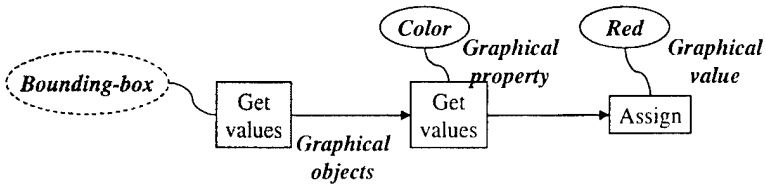
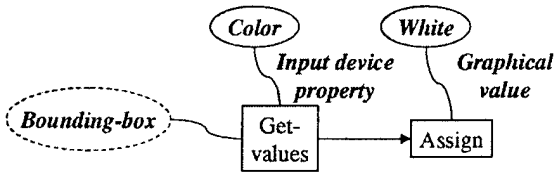
We must decide which specific object definition and transformation functions to use from the broad general classes (e.g. *enumeration, functional description, data transform, mapping transform, graphical transform, rendering transform*) described in the previous chapter. For example in Figure III-1 we use the *assign* function which is a specific instance of the more general graphical transform class. There are no sub-class functions to the enumeration object definition class so no further refinements are needed in that component. We consider this first step as part of the instantiation augmentation process but it can sometimes be included in the functional specification process if specific transform instances (e.g. *assign*) are required to define the "core" functionality of a technique. I.e. a functional specification may consist of broad function classes (e.g. *enumeration, graphical transform*) or of specific function instances (e.g. *assign*). However, a functional specification should be stated in as general and abstract terms as possible. Thus, function instances should only be used when absolutely necessary.

---

[1] The specifications shown in Figure III-1 and in this chapter uses the same notation as the functional specifications (i.e. *ODT* or *object-definition-transformation* diagrams) shown in the previous chapter. Details on the diagrammatic notations used in these specifications can be found in appendix A-1.

[2] In this work we assert that "core" functionality is captured by either object definition or transformation functions and nothing else. Translation functions and input-devices are considered part of the structural (as opposed to functional) aspect of a visualization technique.

[3] Translation functions transform objects between the data and graphical realms and also among the different object types within each realm. More detail on these functions is given in the following sections.

<table>
<tr><td>Functional Specification</td><td> 
</td></tr>
<tr><td>Step 1:</td><td>Pick specific object selection and transformation functions
 
</td></tr>
<tr><td>Step 2:</td><td>Match outputs with inputs through translation functions
 
</td></tr>
<tr><td>Step 3:</td><td>Determine how function inputs are chosen
 
</td></tr>
<tr><td>Step 4:</td><td>Determine input-devices
 
</td></tr>
<tr><td>Step 5: Instantiation Specification</td><td>Specify initialization arguments to the input-devices
<u>Input-device initialization specification</u>: Color property value of bounding-box input-device is set to white.
 

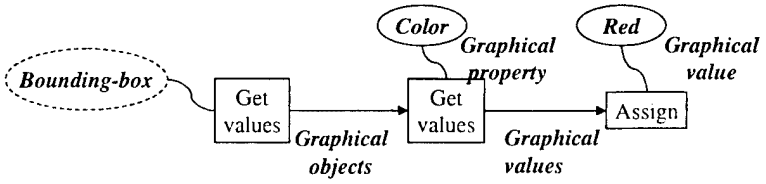
<u>Main technique specification</u>: This is the complete specification from Step 4.
 
</td></tr>
</table>

Figure III-1: Diagrammatic representation of the five-step instantiation augmentation process for the object highlight technique. Additions made in each step are shown in gray.

Step2. Match outputs with inputs through translation functions:

A visualization consists of elements from three different realms, the data realm, the graphical realm and the output media. The transformation and object definition functions within a visualization technique may operate on objects, attributes, and values in any of these realms. Proper operation of a technique depends on whether we apply its object definition and transformation functions to the correct realm and to the correct element types within each realm. A set of *translation functions* is available to translate the outputs of functions and devices between the realms. These translation functions are also used to explore object relationships and query for object state so that we may obtain the correct element types within each realm. At this point in the instantiation augmentation process, we decide which translation functions need to be inserted between the object definition and transformation functions declared in step 1 to ensure that the outputs of the object definition function match the inputs of the following transformation function. For example in Figure III-1, step 2, the *enumeration* object definition step produces a set of graphical objects. We use the *get-values* translation function to extract a set of *color* graphical values from these objects because graphical values are needed in the subsequent *assign* graphical transform.

Step3. Determine how function inputs are chosen:

Object definition, transformation, and translation functions all have inputs and outputs. The input arguments to these functions can be provided by other functions, such as a translation function (step 2), an object definition function or a transformation function (as is outlined in chapter II-2). Input arguments that are not provided by a function must be preset by a visualization designer or controlled by a user through an input-device. Here we decide between these two alternatives. For example in Figure III-1, step 3, the object highlighting technique has three unspecified input arguments after the first two steps. These arguments include: 1) the set of graphical objects to feed into the *get-values* function, 2) a graphical property for the *get-values* function, and 3) a property value for the *assign* transform. In this example, we provide the latter two values through designer defaults. The graphical property is set to *color* and the property value is set to *red*. The first input argument is obtained from the user through an input-device.

Step4. Determine input-devices:

In this step we determine the type of input-devices to use with the interactive system. An input-device is needed for every user input value specified in step 3. We must decide whether to provide several input values with a single device (composition of input-devices) or whether to use separate devices to provide separate inputs. The types of devices that are appropriate would depend on effectiveness measures such as the ones outlined by Card et. al. [*Card, 1990, Mackinlay, 1990*]. In Figure III-1 there is only one user input argument and we use a *bounding-box* that allows users to *enumerate* the set of objects they want to highlight. Note that a *get-values* translation function is used with all input-devices to extract relevant state information from them.

Step5. Specify initialization arguments to the input-devices:
Certain input-devices may require initialization values. For example, a *slider* input-device must be initialized with the maximum and minimum slider range. On the other hand a *scroll list* is initialized with a list of selectable entries. In the object highlight example, the *bounding-box* input-device **does not** require any initialization, however, we decided to initialize its *color* property anyway for aesthetic reasons. Note that the operations used for input-device initializations are visualization techniques themselves. In this case, the object definition component defines a *bounding-box* input-device object that is a designer enumerated object. Subsequently this object is passed to the *assign* graphical transform operator, which changes the *color* property of the *bounding-box* to *white*. For a more detailed example of the structural augmentation process refer to appendix B-3.

To perform the *instantiation augmentation process* outlined above, we must clearly define the specific object definition, transformation, and translation functions available; the set of visualization elements from the three realms (data, graphical, output media) that are manipulated by these functions; and the input-devices that can be used with these functions. In section III-1 we give detailed descriptions of the primitives within these three realms. Section III-1 is not meant as advancement to the field. Many of the visualization elements and input-devices presented have been captured in previous work, and the visualization function primitives recapitulates standard math theory that can be accomplished using any current programming language. However, this level of detail is necessary for our automatic design system because it requires a complete description of all primitives that are available for design. In addition, the primitives also give specific examples of the types of functions we would find in each transformation class and provides bounds on our framework, indicating the number of primitives that are required to capture the various visualization techniques described in this work. Section III-1 may be skipped if the reader is not interested in detailed descriptions of the primitives used by our automatic design system. In section III-2 we show how changing the instantiation specification of a technique can change its effectiveness at solving tasks and can sometimes lead to new and interesting design variations. Note that the design alternatives generated by exploration at the instantiation level is different from that of the functional level (chapter II-3) because here, we are keeping the semantics of a technique constant and only changing its structural content. As a result the technique still fulfills the same goals even though the method of interaction or the visual feedback may now be different. Finally we close the chapter by discussing the merits of our framework based on three criteria, completeness, coverage, and practicality (section III-3). These last two sections (sections III-2, III-3) are provided to validate and highlight the uses of our framework. They may be skipped if the reader is only interested in the automatic design aspects of this work.

III-1 Representation Language

In this section we present the visualization elements, functions, and devices defined in our framework for constructing visualization techniques. These primitives were picked based on common features

available in current visualization methods. It is important to note that this is not a complete list, and it can never be complete because as new techniques are developed the list must necessarily grow and change. The primitives described in our framework, however, are flexible and can be used to attain a variety of visualization effects. Also note that the primitive functions have low granularity (i.e. we use simple mathematical functions). This level of granularity gives us greater flexibility in composition and allows us to express many current visualization techniques with a relatively small set of primitive building blocks. In the next sections we describe the three primary building block classes in our framework:

1) *Visualization elements or properties* which may be from the data realm, graphical realm, or output media. These objects or object properties form the inputs and outputs of the primitive functions within a visualization technique;
2) *Visualization technique primitive functions* which may come from the object definition, transformation, or translation function classes.
3) *Input-devices* that provide users with physical (e.g. *mouse clicks*) and/or virtual (e.g. *menus*) controls so that they may interact with a visualization technique.

III-1.1 Visualization Elements or Properties

Visualization techniques operate over the data concepts, graphical objects, or output media that form a visualization. In the object definition component of a visualization technique a set of these visualization objects is chosen and then transformed. In addition to the elements selected in the object definition component, primitive visualization functions often require other object or value inputs as was illustrated in Figure III-1. These other inputs provide designers with a limited means of controlling the behavior of the functions so that they can achieve a wide range of effects with a relatively small set of functions. In this section we present the representational structures used to describe abstract data concepts and graphical objects within a visualization. Our representational structures are based on previous work [*Mackinlay, 1986a, 1986b; Roth, 1990*]. We do not characterize the output media here because the focus of this thesis is only on data and mapping functions, which operate wholly in the data and graphical realms. We leave characterization of the output media for future work.

III-1.1.1 Data Concepts

The primary element in the data realm is the data concept. A data concept is a database record, very commonly represented as rows in spreadsheet programs. For example in Figure III-2 the data concepts being represented are *houses*. Each data concept is attached to a *data type* that describes the attributes and relational structure of the data concept. Data concepts having the same attributes and relational structure will be attached to the same *data type*. Every *house* data concept or record, for example, belongs to the *house-data-type* class. The *house-data-type* class describes the five *data attributes* that are attached to each *house* data concept, namely *house-selling-price, date-sold, neighborhood, date-on-market,* and *number-of-rooms*. Each data attribute commonly corresponds to a column of values in spreadsheet programs. The first three attributes of the *house* concepts are mapped to graphics in Figure III-2.

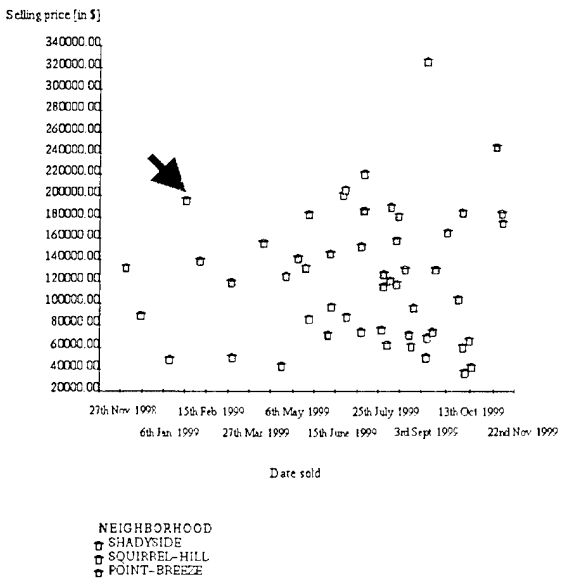


Figure III-2: Example visualization containing house data. Each mark represents a house data concept. The *x-axis* shows the *date-sold* data attribute; the *y-axis* shows the *selling-price* data attribute, and *hue* shows the *neighborhood* data attribute.

Although data concepts commonly represent "real-world" objects, e.g. a house, this need not be the case. A data concept may also represent a conceptual object, consisting of a group of related *data attributes* possibly originating from multiple different "real-world" objects. For example, we may have data concepts that contain the *house-selling-price* attribute as well as the *owner-salary* attribute. In this case the *house-selling-price* is an attribute of a house object whereas the *owner-salary* attribute is an attribute of a person. Depending on the task, our automatic system may draw data attributes from multiple different data concept classes to form new conceptual objects as required.

III-1.1.2 Graphical Objects

In a visualization, data concepts in a database are mapped to graphical objects in a graphical scene. Graphical objects, also commonly referred to as glyphs, are symbols that represent information through visual properties that are either spatial (*position-x, position-y*), retinal (*color, size*), or temporal (*jittering*). Graphical objects may be simple (e.g. *mark, bar*) consisting of only a few properties or more complex (e.g. *Chernoff Faces*[*Chernoff, 1973*], *InfoBug*[*Chuah, 1998a*]) consisting of many properties. Graphical objects commonly live within container objects (e.g. *chart, map*). For example, Figure III-2 shows a house data visualization. The *house* data concepts were encoded using *mark* graphical objects within a *chart* container. Containers are used to structure graphical objects (e.g. *marks*), annotation objects (e.g. *axes, axes-labels*) and other container objects. The *chart* in Figure III-2 for example is used to group the *mark* graphical objects together based on a well-defined layout scheme.

Based on previous frameworks [*Mackinlay, 1986b; Chuah, 1995*], we organize graphical containers and objects into a hierarchy. At the top of the hierarchy is the *visualization container*. The visualization container exists within a desktop window, i.e. all the contents of the window are considered part of the visualization. For example, Figure III-3 shows a visualization of *house* data containing *charts, tables, marks, bars, axes,* and *axis-labels*. Figure III-4 shows a hierarchical breakdown of the components within the visualization in Figure III-3.

Below the visualization container are *region containers*. Regions are arranged based on very specific data attribute constraints. Two regions can be aligned (i.e. laid out side by side either horizontally or vertically) only if their common axes represent the same type of data attributes. Figure III-3, for example, has three aligned regions. Horizontal alignment is allowed here because the common axis of the three regions (*y-axis*) encodes the same data attribute (i.e. *house-address*) in all three charts.

Region containers group graphical objects together and structure them according to a layout scheme. Some example layout schemes are shown in Figure III-5. The *grid* layout, for example, constrains the positions of graphical objects so that they fall on evenly spaced rows and columns. The *chart* layout on the other hand does not have any positional constraints so that the positions of the graphical objects may be used to encode data, as in Figure III-2.

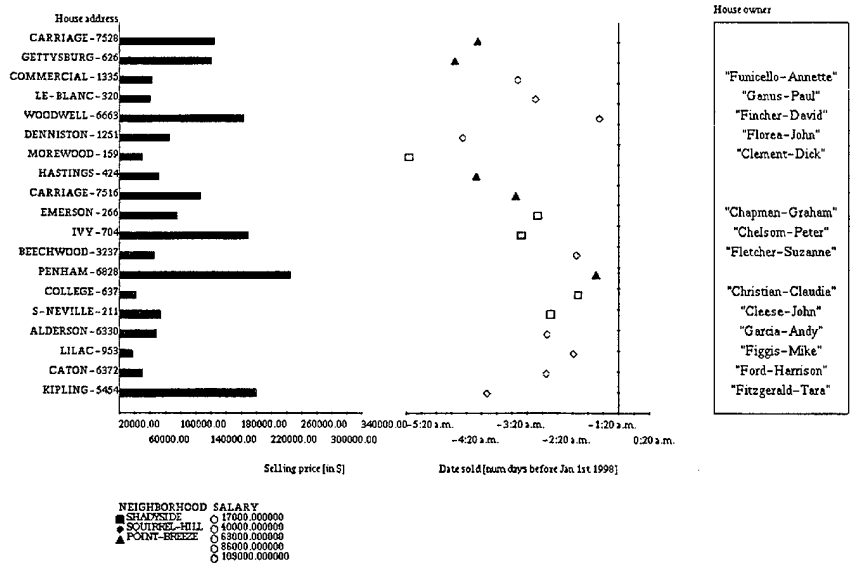


Figure III-3: Example visualization of house data. Hierarchical breakdown of graphical objects in this visualization is shown in Figure III-4. *X-axis* in left-most chart shows *selling-price*; *x-axis* in middle chart shows *date-sold, shape* shows *neighborhood,* and *saturation* shows *salary*; *text* in right-most table shows *house-owner. Y-axes* for all three regions show *house-address* data attribute.

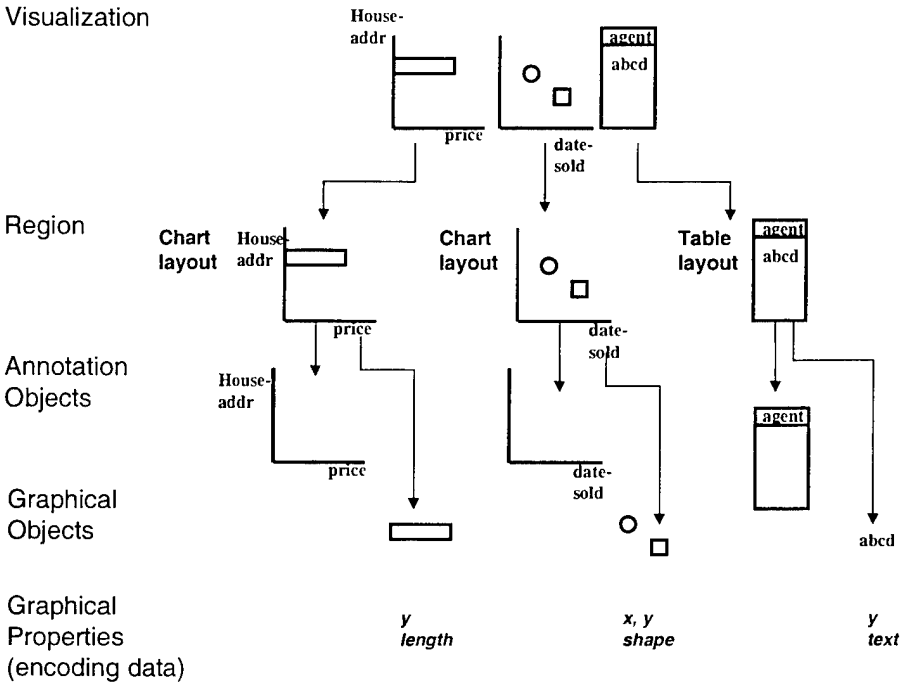


Figure III-4: Container hierarchy for visualization in Figure III-3

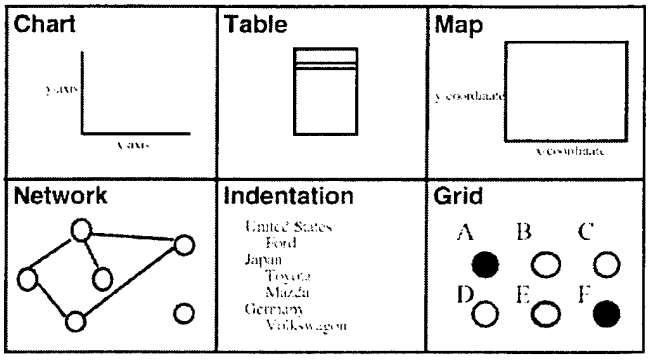


Figure III-5: Example region layout schemes (borrowed from [*Chuah, 1995*])

Annotation objects are a specialized class of graphical objects that are not containers and that **do not** represent data concepts. Some example annotation objects include chart axes, legends, and axis labels. For example, the *chart* in Figure III-2 contains a set of annotation objects including two axes, a set of axis-labels, and a legend indicating how the house neighborhoods relate to the *color* of the graphical *marks*. Annotations are used to present clarification on aspects of the visualization that are not readily apparent to users. For example, annotation objects are commonly used to show how data is encoded using graphics (e.g. *chart-axes* and *legends*), so that we can better interpret the graphical representations and tie them back

to relationships within the data set. Annotations are also very useful for drawing user attention to particular graphical elements and thus are widely used for communicative purposes. For example, the *red* arrow in Figure III-2 is an annotation object for bringing user attention to a particular house object.

III-1.2 Visualization Functions

In the previous chapter we had described the general classes of visualization functions (object definition and transformation). Apart from the object definition and transformation classes, there are also translation functions that are used to convert input and output argument types. In this section we give detailed descriptions of all function primitives available to our automatic design system. Remember that all the functions described here can be accomplished with current programming languages. We enumerate the primitives here because our automatic design system requires a complete characterization of the list of functions that it can manipulate. Listing out these functions is also useful for illustrating the number and type of operators that are sufficient for describing current visualization techniques.

III-1.2.1 Object Definition Functions

In the object definition component we select a set of visualization objects for subsequent transformation. Object set definition can be performed by enumeration or through functional description. Enumeration allows the user or designer to list/enumerate all the interesting visualization objects by name. In contrast, functional description methods allow users or designers to specify an interesting set of visualization objects by applying functional constraints on the objects' attributes or properties.

Function class	**Input**	**Output**
Functional description	1. Value set, 2. Threshold value, 3. Compare operator (>, <, =, >=, <=, <>)	Boolean value set
Set operation	1. *n* object sets, 2. Set operator (*intersect, union, repeat-union, difference*)	Object set

Table III-1: List of object definition functions

Functional description functions may be simple or complex depending on the task requirements. In this thesis we use the *threshold* function as the primary functional description method. Table III-1 shows the inputs and outputs of the threshold function including all the threshold compare operators available (<, >, =, <=, >=). The threshold function has three inputs: 1) a set of input values on which to perform the threshold operation, 2) a threshold value and 3) a compare operator. It then returns a set of boolean values

indicating whether each value in the input set passed the threshold. For example, if we want to find all houses whose price is above *$100k*, we would apply the threshold function to: 1) the set of *house-price* values, 2) the *$100k* threshold value, and 3) the > compare operator. The output will be a set of boolean values, indicating for each input *house-price* value whether the it exceeds *100k*. Note that the *threshold* function may be used to filter any of the elements within a visualization including: 1) data concept attributes such as *house-price* or *date-sold,* 2) graphical object properties such as *x-position* or *size*, 3) annotation object properties such as *size-of-legend, thickness of axes*, and 4) input-device properties such as *bounding-box-color* or *slider-minimum-value*.

The other functions in the object definition component are set operators. *Set operators* compose two or more objects sets together (*object-definition transformation*) to produce a single output set. Set operators are crucial for object definition composition as was illustrated in the previous chapter. Some example set operators include the *intersect, union, repeat-union* and *difference* operators. Below we apply each of these operators to three example object sets and show how their results differ.

difference ({a, b}, {b, c}, {c, d}) = *difference* ({a, c}, {c, d}) = { a, d }
intersect ({a, b}, {b, c}, {c, d}) = {}
union ({a, b}, {b, c}, {c, d}) = { a, b, c, d }
repeat-union ({a, b}, {b, c}, {c, d}) = { a, b, b, c, c, d }

The *repeat-union* operator combines all the input object sets just like the *union* operator except that it does not omit duplicate objects.

III-1.2.2 Transformation Functions

There are four classes of transformation functions, *data, mapping, graphical,* and *rendering* transforms. These four transformation classes correspond to the four main phases of the visualization creation process described in the previous chapter.

III-1.2.2.1 Mapping Transforms

Mapping transforms are the basis for visualizing data because they allow abstract data concepts to be perceived by linking them to visual graphical elements. There are two primary mapping transforms in our framework: *object mapping* and *attribute mapping* (shown in Table III-2).

Object mappings relate a class of data concepts as defined by their data type to a class of graphical objects as defined by their graphical class. Data types capture characteristics of similar data concepts that contain the same data attributes and relationships. Graphical classes capture characteristics of similar graphical objects that have a common visual appearance and contain the same graphical properties. Some common graphical classes include *bar-class*, *mark-class*, *node-class*, and *interval-bar-class*. For example, we can map all *house* data concepts to *mark* graphical objects by mapping the *house-data-type* class to the

mark-graphical-class as in Figure III-2. As is shown in Table III-2, the first two input arguments to the *object mapping* transform are the *data-type* and *graphical class* that we want to attach.

Function class	Input	Output
Add object mapping (Delete object mapping)	1. Data type, 2. Graphical class, 3. Container object or graphical object(s)	
Add attribute mapping (Delete attribute mapping)	1. Data attribute, 2. Graphical property, 3. Container object or graphical object(s) 4. Mapping effect (*forward, backward, both*)	
Add object (Delete object)	1. Data concept(s) 2. Container object	

Table III-2: List of mapping transformation functions

The third object mapping input specifies the scope of the mapping function. The scope is defined by listing the container object (e.g. *visualization container, graphical space, region*) we want to attach the mapping to. Applying a mapping transform to a container object will cause all other containers encapsulated within it (based on the object hierarchy in Figure III-4) to inherit the mapping as well.

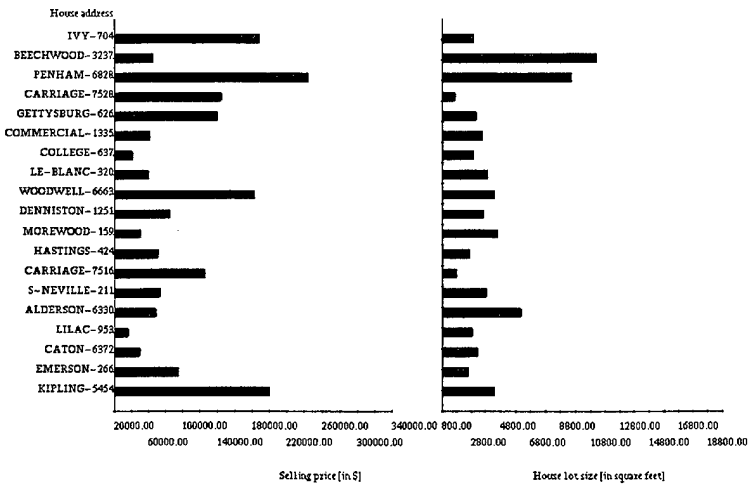


Figure III-6: *House* data-type to *bar* graphical-class mapping applied to the entire visualization. Both chart regions within the visualization inherit this mapping relationship. The *x-axis* of left chart shows *selling-price*; *x-axis* of right chart shows *house-lot-size* and *y-axis* of both charts show *house-address*.

For example, applying an *object mapping* function that maps *house-data-type* to the *bar-graphical-class* onto a *visualization* container will cause all the *region* containers within the *visualization* container to inherit that mapping (e.g. Figure III-6). I.e. all *house* data concepts associated with each *region* container in Figure III-6 gets mapped to *bar* graphical objects based on the mapping transform attached to their parent *visualization* container. Alternatively, we can apply separate object mappings to each of the regions in Figure III-6 instead of just applying one object mapping to the entire visualization container. For example in Figure III-7 we have applied a *house-data-type::bar-graphical-class* object mapping to the left region and a *house-data-type::mark-graphical-class* object mapping to the right region.

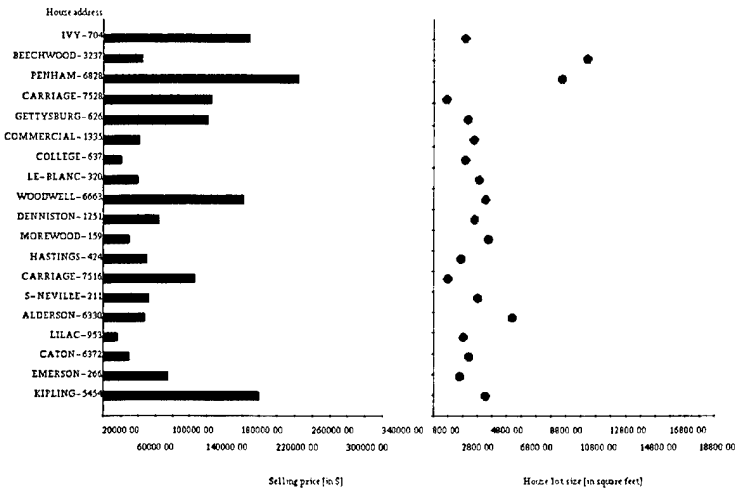


Figure III-7: The same visualization design as Figure III-6 except that a *house* data-type to *bar* graphical-class mapping is applied to the left region and a *house* data type to *mark* graphical-class mapping is applied to the right region.

Mappings very commonly get attached to container objects (e.g. *visualization container, space container* or *region container*) as was just discussed. However, limiting object mapping transforms solely to container objects, only allows us to control how data concepts are mapped to graphical objects on a region by region basis (i.e. we cannot associate subsets of data concepts within a given region to different mapping transforms). In the Table Lens [*Rao, 1994*] technique, the data concepts that appear within the table lens are mapped to the *text* graphical class, and all other data concepts are mapped to the *bar* graphical class. The objects within the *lens* may span multiple column *regions* and only include a subset of objects within each region. This Table Lens operation therefore cannot be achieved with a *region* scope mapping transform To enable Table Lens type mapping, we allow the mapping functions in our framework to be applied to container objects as well as to particular graphical objects within those containers. Note that in order for there to be graphical objects in the first place, we must begin by applying an object mapping transform to a container object. We can then refine the appearance of particular graphical objects within the container by remapping them to a new graphical class. For example in Figure III-8 we have

remapped some of the graphical objects in the left-*region* container of Figure III-6 to the *mark* graphical class.

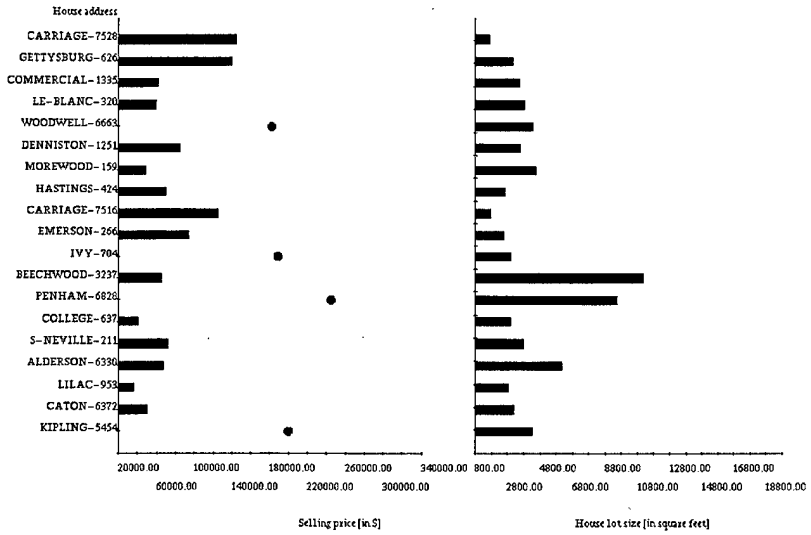


Figure III-8: The same visualization design as Figure III-6 except that a *house* data-type to *mark* graphical-class mapping transform is applied to particular graphical objects in the left chart including *Woodwell-6663, Ivy-704, Penham-6828,* and *Kipling-5454.*

The second primary mapping function is the *attribute mapping* function. Attribute mappings are used to express data attribute values through the use of graphical properties. All visualizations contain a set of data attribute (e.g. *net-profit*, *number-of-rooms*) to graphical property mappings (e.g. *position, color*) for illustrating the trends and relationships of data values visually[4]. As is shown in Table III-2, attribute mappings have four input arguments. The first two arguments are the data attribute and graphical property we want to link. The third argument specifies the scope of the attribute mapping. The scope of an *attribute mapping* is specified in the same way as the scope of an *object mapping*. I.e. *attribute mappings* can be applied to container objects or to graphical objects. Finally the last argument (the *mapping-effect* argument), indicates whether the mapping function allows subsequent changes in data values to affect their related graphical values and vice versa. There are four types of mapping-effect operators: *forward, backward, both* or *none*. *Forward* allows subsequent changes in data values to be reflected in their corresponding graphical values. *Backward* allows subsequent changes in graphical values to be reflected in

---

[4] *Attribute mappings* usually map data attribute values to graphical property values linearly. However, in some cases we may need to use a non-linear function or a modified linear function to take into account peculiarities of the human visual system. For example Teghtsoonian [*Teghtsoonian,* 1965] found that the perceived *area* is typically the actual area raised to a power of .8. Similar discrepancies arise in *length* and *diameter* judgements. We currently do not deal with capturing these different attribute mapping functions in our framework but such extensions would not be overly difficult to implement.

their corresponding data values. *Both* refer to a combination of *forward* and *backward* effects and finally *none* does not allow updated data or graphical values to propagate either way. For example, we may use the *backward* mapping-effect to link the size of a mark to the *selling price* data attribute. This allows us to change the underlying data (i.e. the *selling price* attribute values) by controlling their corresponding graphical representations (i.e. by changing the size of the *mark* graphical objects).

In addition to *object* and *attribute mapping* functions, the mapping transform class also includes the *add-object* and *delete-object* functions. These functions allow us to change the scope of preexisting object and attribute mappings by changing the data concepts that a container object is associated with. The *add-object* transform **associates** a given set of data concepts with a container object and the *delete-object* transform **disassociates** a given set of data concepts from a container object. Associating new data concepts to a container object will cause those data concepts to be added to each sub-container within the container object. The added concepts will then be mapped according to the mapping transforms associated with the lowest container class. For example attaching a new set of houses to the visualization container in Figure III-7 would cause those new data concepts to be mapped to *bars* in the left region and *marks* in the right region. Furthermore, the new data concepts will have their *house-selling-price* data attribute mapped to the *length* graphical property in the left region, their *date-sold* data attribute mapped to the *mark-x-position* in the right region, and their *house-address* data attribute mapped to the *y-axis* in both regions.

III-1.2.2.2 Data and Graphical Transforms

Unlike mapping transforms, which change elements of one class (i.e. data), to another (i.e. graphical), data and graphical transforms change elements of a single class to different forms within that class. There are five classes of data and graphical transforms in our framework (shown in Table III-3): *unary-functions, binary-functions, summary-functions, assign-function* and *specialized-functions*. These data and graphical transform functions are applied to change existing data and graphical values or to generate new values. There are three primary ways in which these transforms are applied:

1. To change or summarize the values of a single data attribute or graphical property.

$$G_{att1} \rightarrow G_{att2}$$

$$D_{att1} \rightarrow D_{att2}$$

2. To convert one type of attribute or property to another.

$$G_{att1} \rightarrow G_{att2}, \quad att1 \neq att2$$

$$D_{att1} \rightarrow D_{att2}, \quad att1 \neq att2$$

3. To derive a new attribute or property based on multiple existing attributes and properties.

$$G_{att1}, G_{att2}, \ldots, G_{att_n} \rightarrow G_{att_m}$$

$$D_{att1}, D_{att2}, \ldots, D_{att_n} \rightarrow D_{att_m}$$

for any n and m, where $n > 1$, and att_m is a new attribute

1. *Change the values of a single data attribute or graphical property*

Example techniques that fall within this first class include most of the operations in the SDM [*Chuah, 1995b*] system that are used to improve the readability of a visualization.

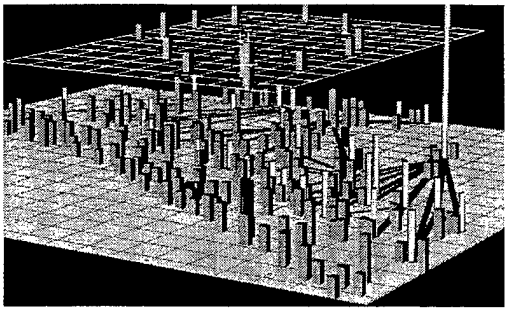


Figure III-9: SDM lift objects technique

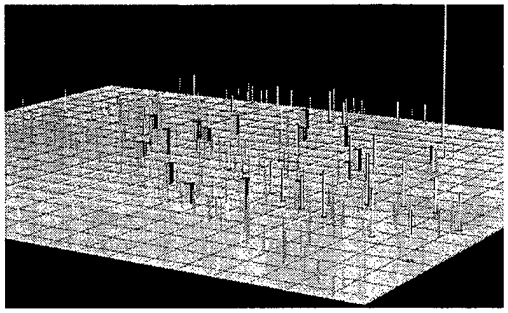


Figure III-10: SDM thin objects technique

Some examples include the *lift objects* technique that allows users to lift selected objects over and above the other objects in the graphical scene so that they are visually more salient and less occluded (shown in Figure III-9). This *lift objects* technique is a $G_{z_position} \rightarrow G_{z_position}$ graphical transform; it changes a set of *z-position* graphical values by adding a constant to it. Another example SDM technique that falls into this class is the *thinning objects* technique that allows users to change the width of contextual graphical objects so that they do not occlude the focus objects (shown in Figure III-10). This example is a $G_{width} \rightarrow G_{width}$ graphical transform. Other examples include the feedback operations in the dynamic query [*Ahlberg, 1992*] and painting techniques [*Becker, 1987*], which is a $G_{color} \rightarrow G_{color}$ transform. Such feedback operations commonly use the *assign* graphical transform to set color values of focus objects to a salient constant value. All the transform functions described above change the values of a single graphical property by adding or subtracting constants to/from the value set or by setting the value set to a constant.

2. *Convert one type of attribute or property to another*

A slightly more complex use of graphical and data transforms is to apply the values of one attribute or property to determine the values of another. A common application of this class of functions is to do value conversions. For example the data attribute *temperature in Kelvins* can be used to calculate the data attribute *temperature in Fahrenheit*, e.g. $D_{Kelvins} \rightarrow D_{Fahrenheit}$.

Another application of this class of transforms is to link properties so that they will change in tandem. For example we may want our rectangle-shaped marks to always appear as squares. In order to achieve this we may map $G_{x_range} \rightarrow G_{y_range}$ using the *assign* function. Once we have done this, any manipulation function that causes the *x-range* property to change will cause a similar change in the *y-range* property.

This class of transforms is also an integral part of expressing animation. Animation is achieved by linking physical time (G_{time}) to a graphical attribute. For example, in order to make the size of marks in a display increase with time, we define a $G_{time} \rightarrow G_{size}$ transform.

Other example techniques that fall within this category include summarization techniques. These techniques are used to summarize or aggregate data or graphical elements so that the visualization is simplified and users can interpret the relationships shown by the graphic design more quickly and effectively. For example, instead of representing all the *house-selling-price* values in our house database, we can group the house concepts by *neighborhood* and show the average/mean *house-prices* by neighborhood ($D_{house_price} \rightarrow D_{mean_house_price}$). In the *PAD++* system, file objects are summarized or aggregated together (i.e. summarized) when there is very little space allocated to them ($D_{file} \rightarrow D_{aggregate_file}$). When users *zoom* in on an aggregated file object, it gets split up into its individual components.

The final operations that fall within this category are specialized functions that are used to extract meta-data from a set of values. Some examples include determining the order of the values (e.g. *sort* transform: $D_{attribute} \rightarrow D_{sort}$) or the number of times a particular value appears (e.g. *count*: $D_{attribute} \rightarrow D_{count}$). This meta-data allows users to analyze additional structural information about the existing data set.

3. Derive a new attribute or property based on multiple existing attributes and properties

Data and graphical transforms may also be used to calculate derived values from multiple attributes and properties. For example we can determine the *gross profit* of various *company* data concepts *by subtracting* their *total-cost* attribute values from their *total-sales* attribute values. In this case a new data attribute, *gross-profit* is generated based on existing data attributes *total-cost* and *total-sales* (D_{total_sales}, $D_{total_cost} \rightarrow D_{gross_profit}$). Another example is the *SDM-distance* technique where the *x-position* and *y-position* of objects are used to calculate their distance to the user ($G_{x_position}$, $G_{y_position} \rightarrow G_{distance}$). Note that this class of functions is similar to the previous one except that we have multiple attributes resulting in only one attribute (i.e. a *many-to-one* mapping in contrast to the *one-to-one* mapping of the previous section). Thus similar to the previous category we can convert, link, and summarize attributes and properties, as well as compute meta-data.

In Table III-3 we list the data and graphical transform instances in our framework as well as their input and output arguments.

Function class	Function name	Input	Output
Unary Function		1. Unary operator, (*complement, absolute*) 2. Single value set.	Single value set
Binary Function		1. Binary operator, (*add, subtract, multiply, divide*) 2. *n* value sets.	Single value set
Summary Functions	Group objects	1. Single object set, 2. Group object data type.	Group object
	Summarize values	1. Summary operator, (*sum, mean, median, std-deviation, min, max*) 2. Single value set.	Single value
Assign Functions	Assign	1. Single value set containing the destination values, 2. Single value set containing the source values.	
Specialized Functions	Sort	1. Sort operator, (*increasing, decreasing*) 2. Single value set.	Single value set
	Count	1. Single value set containing the values we want to count, 2. Single value set that is being counted.	Single value set

Table III-3: List of data and graphical transformation functions

1. Unary functions

Unary functions take a value set as input and produce a transformed value set. We consider two types of unary operators: *complement* and *absolute*.

The *complement* operator may be applied to:

1) numbers (i.e. quantitative and discrete data attributes)
   (e.g. –10 → 10, or 5.23 → -5.23).
2) boolean values
   (e.g. T → F, F → T)
3) strings
   (e.g. abc → cba, dracula → alucard)

The *absolute* operator may be applied to:

1) numbers
   (e.g. –10 → 10, or 5.23 → 5.23).

2. Binary functions

Binary functions, unlike unary functions, take n sets of values (where $n > 1$) and a binary operator as input. The binary operator is applied to each value set in turn, e.g. $n_1 + n_2 + \ldots + n_m$, where n represents a particular value set and + is the input binary operator. There are four binary operators in our framework: *add, subtract, multiply, divide*. The effects of each operator are shown below.

The *add* operator may be applied to:
numbers
$(x, y) \rightarrow x + y$
strings
("abc", "bc") → *"abcbc"*

The *multiply* and *divide* operators may be applied to:
numbers
$(x, y) \rightarrow x * y$ or $(x, y) \rightarrow x / y$

The *subtract* operator may be applied to:
numbers
$(x, y) \rightarrow x - y$
strings
("abc", "bc") → *"a"*

3. Summary functions

In addition to binary and unary compute operators, there is also a set of summary operators (third row of Table III-3). Very often, especially in large data sets, we may want to summarize a set of data concepts and represent the set as a single summary object in order to reduce clutter. This is done by the *group objects* transform which creates a group data element from a set of data concepts. Like the *group objects* operator, the *summary compute* operator is also used for summarization. However, it summarizes a set of values instead of a set of objects. For example, the summary compute operator was used in the *modified value painting* technique (chapter II-2.3) to calculate the mean *house-price* value which is then used in a subsequent object definition threshold function.

4. Assignment function

Unary, binary, and summary functions produce new data and graphical values. To update existing data attribute or graphical property values with new values we use the *assign* function. The *assign* operator takes two value sets as input and assigns the second value set to the first value set. For example suppose we want to update the *house selling-price* data attribute in our database by adding in a new house sales tax value. We can do this by first computing the new *house-selling-prices* using the *addition* and *multiplication* binary operators and then assigning the new values to the old house price attribute values with the *assign* operator.

5. Specialized functions

Finally there are a set of specialized data and graphical transform functions. These functions correspond to common statistical computation operators: *sort* and *count*. The *sort* transform is used to produce a set of ranks based on the numerical, alphabetical or semantic ordering of the input value set. The *count* transform determines the number of times a particular element occurs in an input set. We can either count all the existing elements in the input set or only specific chosen elements.

It is important to note that even though data and graphical transforms use the same functions to transform objects and values, their end goals are very different. Data transforms are used to prepare data concepts and values in a way that is appropriate for our task(s). As described in the previous chapter, data transforms are used for three primary purposes: 1) to calculate derived results, 2) to summarize existing data, and 3) to compute meta-data based on existing information. Graphical transforms, however, are used to provide feedback and to improve the readability of a visualization whose contents are already defined by the data and mapping transform stages.

III-1.2.2.3 Rendering Transforms

Rendering transforms are used to map a graphical scene onto an output media such as the CRT screen. Currently the only primitive rendering function in our system allows us to access the camera in the graphical scene and change the camera parameters such as *position, rotation, focal length,* etc. This allows us to navigate (pan, zoom, rotate) within any visualization that is generated. We have left out detailed descriptions of rendering functions because this thesis is only focussed on the use of data processing and mapping functions. For information on distortion rendering techniques refer to Leung et al.'s taxonomy [*Leung, 1994*].

III-1.3 Input & Output Translation Functions

Apart from picking specific object definition and transformation functions, we must ensure that the arguments of a source function match the arguments of a destination function as was described in step 2 of the instantiation augmentation process[5] outlined earlier in this chapter. To achieve this, there are translation functions that allow data and graphical elements to be queried for related attributes, properties, and relationships. For example we may query a set of data concepts for the set of graphical objects that are used to represent them, or we may query a visualization for the set of data concepts associated with it. Table III-4 shows these translation functions.

---

[5] The instantiation augmentation process is the five step process for converting functional specifications into instantiable visualization techniques.

Translation function class	Function name	Input	Output
Data & Graphical object translation functions	Get related graphics	Single set of data concepts	Single set of graphical objects
	Get data type	Single set of data concepts	Single set of data types[6]
	Get data concepts	A data type, or a container object	Single set of data concepts
	Get related data	Single set of graphical objects	Single set of data concepts
	Get graphical class	Single set of graphical objects	Single set of graphical classes[7]
	Get graphical objects	A graphical class, or a container object	Single set of graphical objects
Container object translation functions	Get parent	A graphical object or a container object	A container object
	Get children	A container object	A set of container objects
	Get mapped data attributes	A container object	A set of data attributes that are mapped to graphics within the container
	Get mapped data types	A container object	A set of data types that are mapped to graphics within the container
	Get mapped graphical properties	A container object	A set of graphical properties that are mapped to data within the container
	Get mapped graphical classes	A container object	A set of graphical classes that are mapped to data within the container
Object attribute translation functions	Get object attributes	A visualization object (e.g. a data concept, a graphical object, a data type, a graphical class, a container object, or an annotation object)	A set of attributes
Attribute value translation functions	Get values	1. A set of visualization objects 2. An object attribute	A set of values
Value translation functions	Get objects	A set of values	A set of visualization objects
	Get boolean objects	1. A set of visualization objects 2. A set of boolean values	A set of visualization objects
	Get named object	1. A string	A visualization object
	Get type	A visualization object	A string
System wide translation functions	Get all objects	Object type (e.g. data concept, graphical object, visualization, region, etc)	A set of objects

Table III-4: Input and output translation functions

Note that all translation relationships can be queried both ways. For example being able to query for all the graphical objects associated with a set of data concepts (*get-related-graphics)* means that there is a related translation function that allows us to query for all the data concepts associated with a set of

[6] Data types capture characteristics of similar data concepts that contain the same data attributes and relationships.

[7] Graphical classes capture characteristics of similar graphical objects that have a common visual appearance and contain the same graphical properties. Some common graphical classes include *bar-class*, *mark-class*, *node-class*, and *interval-bar-class*.

graphical objects (*get-related-data*). Or being able to query a data concept for its related data type (*get-data-type*) means that there is a related translation function for querying a data type to get all the data concepts associated with it (*get-data-concepts*). This symmetry allows us to easily move back and forth between the data and graphical realms as well as between different object types within each realm so that we can flexibly build a wide range of visualization techniques using transformation functions that are in any order.

Object translation functions allow us to query a set of data concepts for their related graphical objects and vice versa. We may also query data concepts for their data type or for the attributes they contain (i.e. *get-attribute-values* function). Similarly we may access the graphical class and properties of graphical objects.

Container functions allow us to query a container for other container objects based on the hierarchical relationships described in section III-1.1.2 (through the *get-parent* and *get-child* functions). In addition, the *get-graphical-objects* and the *get-data-concepts* functions (listed in the *data & graphical object translation function class*) allow us to access all the data or graphical objects associated with a container. We may also query container objects for all the data attributes, data-types, graphical properties and graphical classes that are currently involved in an object or attribute mapping (e.g. *get-mapped-attributes, get-mapped-data-types, get-mapped-properties* and *get-mapped-graphical-classes*).

Object attribute and attribute value translation functions allow us to query data concepts, graphical objects, and container objects (e.g. *visualization, graphical space, region*) for the set of attributes and values associated with them. For example a *house* data concept may contain the *house-price* and *date-sold* attributes, a *mark* graphical object may contain the *x-position* and *color* attributes, and a *visualization* container may have the *size, x-position* and *y-position* attributes. These attributes and values can be extracted from their corresponding objects by using the *get-object-attributes* and *get-values* translation functions.

Value translation functions allow operations on value arguments. For example, values can be queried using *get-objects* value translation functions to obtain the objects containing those values. The *get-boolean-objects* function filters an input object set based on a set of **boolean values**. Specifically, only those objects that have a corresponding *True* value in the boolean value set are included in the function output. The *get-named-object* function returns the visualization object that corresponds to the input **value string**. The *get-type* function returns the class to which an object belongs (e.g. *data concept, graphical object, visualization object,* etc).

Finally there is a set of global translation functions that allow us to query for system wide state such as getting all the data concepts or graphical objects within the entire visualization system (e.g. *get-all-data-concepts, get-all-graphical-objects*). Similarly we may also query for all existing data types, graphical classes, regions, spaces, and visualization containers.

III-1.4 Input-devices

To characterize input-devices, our framework uses Foley et al.'s [*Foley, 1990*] description of input-devices, which consists of three levels of design: *lexical design*, *syntactic design* and *semantic design*. *Lexical design* refers to how input primitives are derived from basic hardware functions. Input primitives include all physical device signals such as mouse clicks, key presses, etc. *Syntactic design* consists of a set of rules by which primitive input units can be composed or joined to form ordered sequences of inputs. For example a series of mouse movements, mouse clicks, and mouse releases are required for specifying the syntactic design for a *bounding-box*. Devices that are built from a well-defined sequence of physical device signals are also referred to as *virtual devices*. Note that while syntactic constructs describe how a device may be manipulated, they do not define its meaning (i.e. its semantics). *Semantic design* defines the meaning of a syntactic construct. For example, *mouse clicks, bounding-boxes* and *sliders* can all be used to define a *selection* of objects. In this case the semantics of the device is the selection of objects, while the actions used to achieve this selection could take multiple syntactic forms (i.e. *mouse clicks, bounding-boxes* or *sliders*). Similarly, a syntactic form can have several meanings. For example a *bounding-box* can be used for selecting a set of objects or for defining a set of values, one at each of its vertices. The list of input-devices considered in this thesis are listed in Table III-5.

Input-device	**Input-device trigger event (syntactic)**	**Required initialization attributes (semantic)**	**Output arguments (semantic)**
Bounding-box	Mouse up	Mouse button that activates the device (either *left, middle,* or *right*)	Annotation objects, Graphical objects, Region, Vertex values
Mouse click	Mouse up	Mouse button	Annotation objects, Graphical object, Region
Option menu, Scroll list, Radio buttons	Double click	A set of strings to put into the device	A single or set of strings
Text box	Enter key	Label	A string
Button	Mouse up	Label	A boolean value
Slider, Dial	Mouse move	Minimum and maximum range of device	One or more values

Table III-5: Input-device Query functions

To define an input-device in our framework, we must define it in terms of the three levels of design described above (physical, syntactic, and semantics). First of all, we define the types of input signals available. In this thesis we only consider input from two physical devices, the *mouse* and the *keyboard*. At the syntactic level, we define the input-device's appearance (i.e. menu, slider, etc) and the input primitives used to control it. This includes the *trigger event*, i.e. the physical event that triggers an update of the input-device. For example *double clicking* would update a *menu*, or a *mouse release* event would update a *bounding-box*. Finally at the semantic level, we define all the initialization arguments needed, and all the output arguments the device is capable of producing. For example, in a slider device class there are two important initialization attributes: the min and max values of the slider. The slider can then be queried for the value(s) marked within it.

III-1.5 Summary

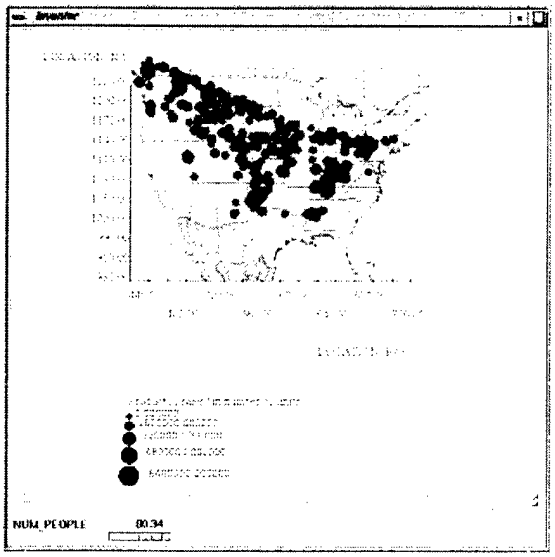
In this section we presented detailed lists of all the visualization objects, functions, and input-devices our automatic system may operate on. We mainly focussed on characterizing data and mapping transforms because the focus of this thesis is on integrating data processing and mapping functions. We also described some graphical transforms because the primitive computations they use are identical to data processing operations (section III-1.2.2.2). We left most of the rendering functions to be specified in future work. It is important to reiterate that the list of argument types, functions, and input-devices provided in this section is not complete. The functions included were chosen because we felt that they were effective for capturing the functions of current information visualization techniques. Now that we have presented all the primitive operators within our framework, we can also them to systematically explore the instantiation space of interactive techniques (sections III-2 and III-3). These sections may be skipped if the reader is only interested in the automatic design aspects of this work.

III-2 Visualization Techniques Instantiation Space

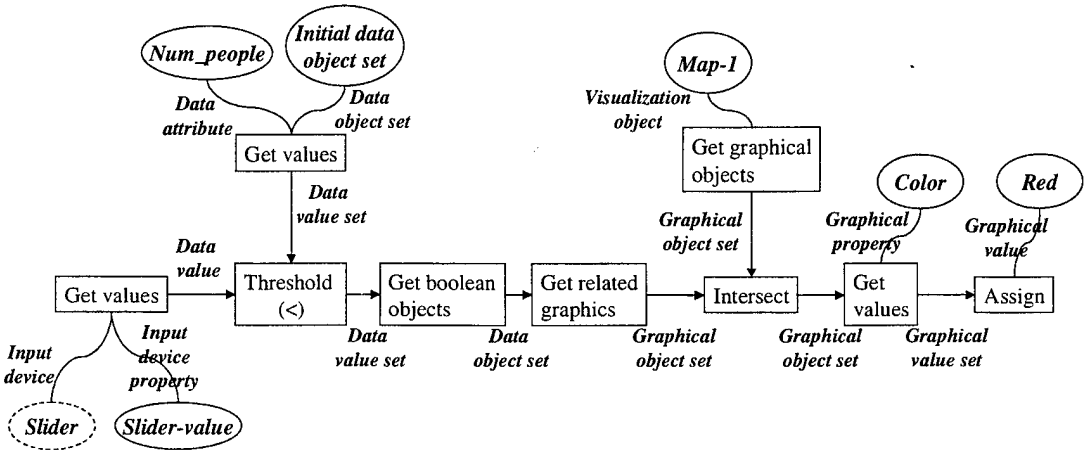
At the end of the previous chapter we show how our framework can help us explore new techniques within the functional design space and improve on existing techniques by combining their functionality. In this section we explore the instantiation design space. Unlike exploration in the functional space which may change the semantics of a technique, exploration in the instantiation space only changes the structural aspects of the technique such as how the technique gets manipulated, which aspects of the technique users get to control, what type of feedback is used, etc. Changes to the instantiation description **do not** change the semantics of a technique. In this section we present an example visualization technique and show how making changes to its instantiation specification can improve its effectiveness. We explore the instantiation space in greater detail in appendix B-4 where we systematically lay out all the alternative instantiation designs for the dynamic query slider technique. In appendix B-5 we explore the instantiation design space for a set of current visualization techniques.

In this example we analyze data from a set of distributors for a hypothetical company in the *United States*. Each mark on the maps in Figure III-11, Figure III-12, and Figure III-13, represents a distributor. The *location* of a mark encodes the geographic location of a distributor and the *size* of a mark encodes the total number of *product-X* units sold by that distributor.

At the bottom of the interface in Figure III-11a is a *dynamic query slider* technique that allows users to highlight distributors based on number of employees (*num_people*). This is done by marking a *num_people* threshold value on the *slider*. Subsequently, all distributors whose *num_people* exceed the marked threshold will get highlighted *red*. The slider visualization technique described above can be defined by the instantiation specification in Figure III-11b. In this specification, we query the slider input-device for the threshold value marked within it. This value is piped into a *threshold* function that filters all of the *num_people* data values. The *threshold* function returns a set of boolean values that we convert back to data concepts using the *get-boolean-objects* translation function. At this point we have a set of distributor concepts whose number of employees are below the threshold value. We query these distributor data concepts for all the graphical objects used to represent them. These graphical objects are then intersected with the graphical objects in the map visualization. The intersection operation is necessary to localize the highlight effect to only the map visualization. Finally the intersected set of graphical objects are colored *red* using the *assign* graphical transform operator.



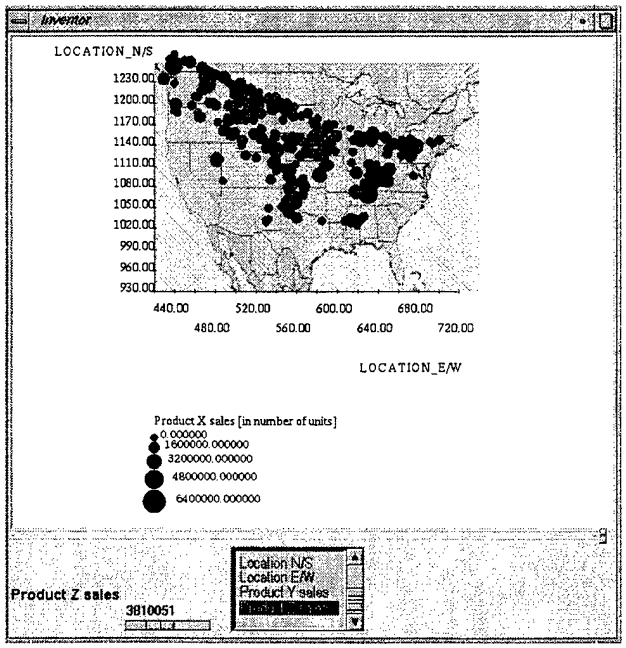
(a) Each mark in the map encodes a *distributor* for a hypothetical company. The *x-axis* and *y-axis* encodes the geographic location of the *distributor*. *Size* encodes the total number of units sold for *product-X*. The slider at the bottom of the interface allows users to select *distributors* based on the number of employees (*num_people*) there. Selected *distributors* are highlighted in *red*.



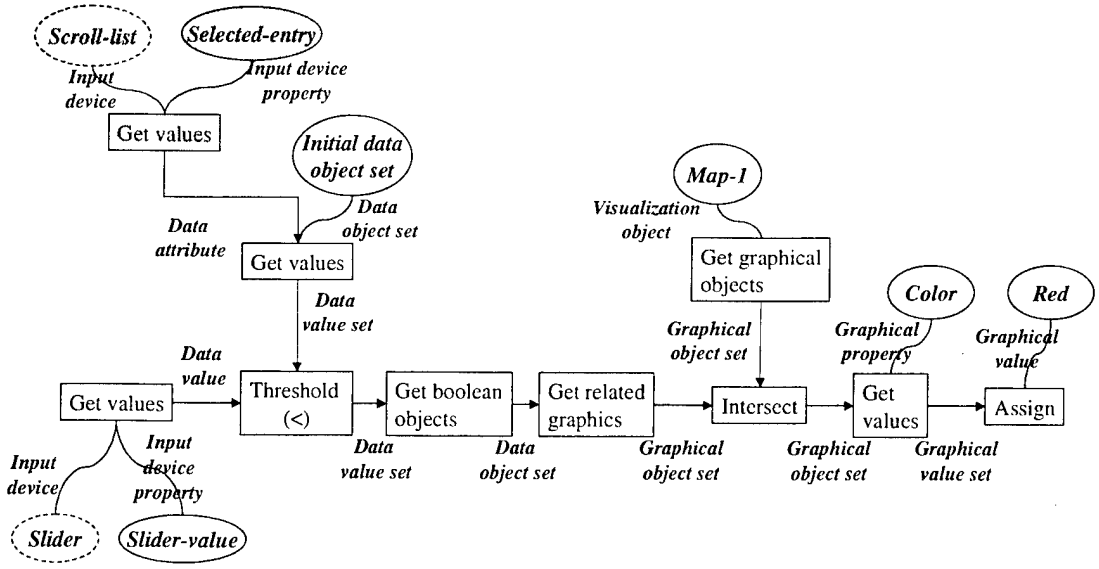
(b) Instantiation specification for the dynamic query slider technique in Figure III-11a

Figure III-11: Example of a dynamic query slider technique that allows users to select various *distributor* data concepts based on the number of employees (*num_people*) working at each site.

A problem with the visualization technique in Figure III-11 is that users can only highlight distributors based on their *num_people* attribute. In order to relax this constraint, we add a new input-device as shown in Figure III-12.



(a) Similar interface to Figure III-11a except that here users get to control a *scroll-menu* in addition to the *slider* in order to pick the current constraint data attribute. Currently, the *product_Z_sales* attribute has been picked on the *scroll-menu* and therefore it appears as the constraint attribute next to the *slider*. Therefore the highlighted objects are those *distributors* with *product_Z_sales* less then 3810051 units.



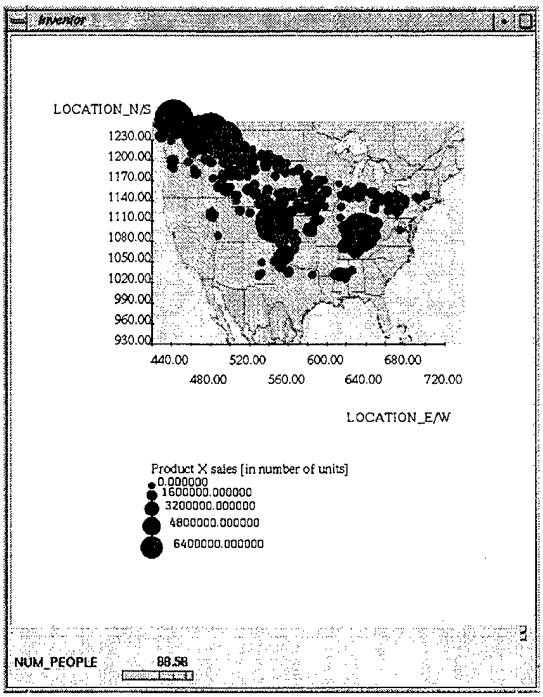
(b) Instantiation specification for the visualization interface in Figure III-12a. **This specification is similar to the one in** Figure III-11b **except that a *scroll-menu* has been added to allow users to pick the constraint attribute. Changes made to the specification in** Figure III-11b **are shown in gray here.**

Figure III-12: Example dynamic query slider technique with selectable data attribute constraint

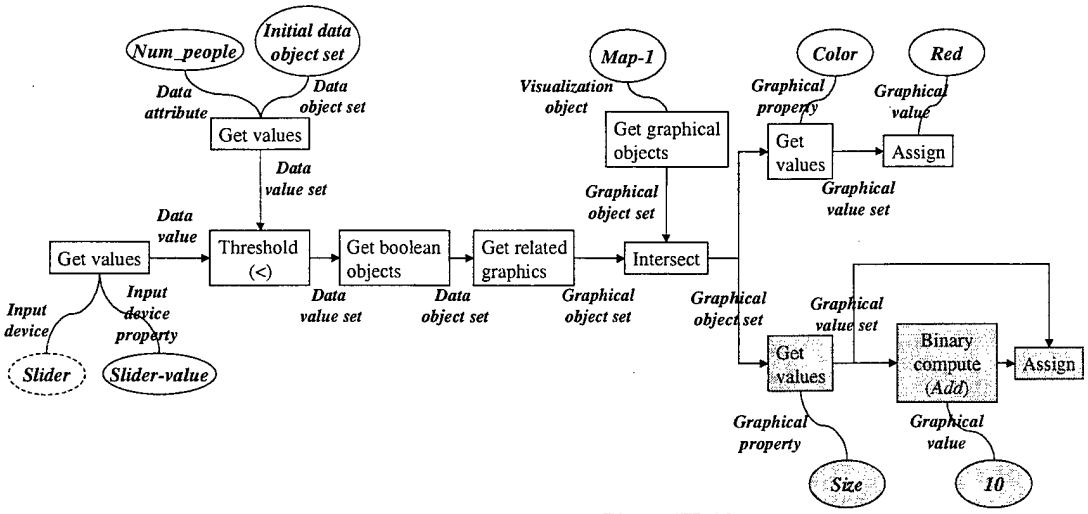
In Figure III-12b we have added a *scroll-menu* input-device which allows users to supply the data attribute on which to perform the threshold operation. Alterations made to the specification in Figure III-11b are shown in light gray in Figure III-12b. The resulting interface (Figure III-12a) is identical to Figure III-11a except that there is a *scroll-menu* below the *slider*, which contains all the distributor attributes (e.g. *location-n/s, location-e/w, product-X-sales, product-Y-sales, product-Z-sales*, and *num_people*). Through this interface users can not only pick the threshold value but also the threshold attribute.

Another problem with the slider highlight technique in Figure III-11 is that occasionally, object *coloring* alone does not provide sufficient feedback because the colored objects may be occluded, thereby making them difficult to find in the visual display despite their coloring. One way to solve this problem is to both enlarge the objects as well as color them. In this way, the highlighted objects become much more salient. We do this simply by adding some new graphical transform operators to the technique specification in Figure III-11b (changes are shown in light gray in Figure III-13b). This new specification will cause the selected objects to be colored *red* as well as enlarged as is shown in Figure III-13a[8].

[8] Note that enlarging the selected objects is sometimes undesirable because they tend to occlude each other or the objects around them. We can alleviate this problem somewhat by changing the draw order of the objects so that the smaller elements are drawn last.



(a) Similar interface to Figure III-11a except that here the selected objects are both colored *red* as well as enlarged in order to increase saliency.



(b) Instantiation specification for the visualization interface in Figure III-13a. This specification is similar to the one in Figure III-11b except that some additional graphical transform functions are included to increase the size of the selected objects. Changes made to the specification in Figure III-11b are shown in gray here.

Figure III-13: Example dynamic query slider technique with color and size feedback on the selected objects

The examples in this section show that refining a visualization technique at the instantiation level allows us to improve the effectiveness with which a technique can be applied to the current task situation and to our current preferences. This section also illustrates some of the design decisions that must be made

by our automatic designer and by human designers when creating a visualization technique. We show in appendix B-4 that by following the five steps in the instantiation augmentation process refinements to the instantiation specification can be carried out systematically and effectively. In appendix B-5 we begin analyzing the instantiation space for a set of current visualization techniques. These examples further illustrate the differences between the functional and instantiation levels from a design standpoint and show the applicability of our two level design methodology. Specifically, exploration in the functional space is useful when we want to create new techniques that have unique or additional functions. On the other hand exploration of the instantiation space is useful when we want to refine a technique to better suit the current hardware, user preferences or task conventions, without altering its underlying functionality. Note that the customization examples shown in this section (Figure III-11, Figure III-12, Figure III-13) do not alter the general functionality of the slider technique. The function of the slider technique, which is to allow users to select a set of data concepts, based on their attributes and then to highlight their corresponding graphical objects, remains the same in all three specifications.

III-3 Evaluation of Framework

We evaluate our framework based on three criteria: completeness, coverage, and practicality. *Completeness* refers to whether the framework is capable of expressing all visualization techniques. *Coverage* refers to whether the framework can be applied to a wide range of visualization types (e.g. bar charts, scatter-plots, 3D-displays) and input-devices (e.g. *menus*, *bounding-box*, *radio-buttons*). And finally we argue for *practicality* in three ways: 1) the framework reduces the cost of task tailoring; 2) the framework provides a new design methodology; and 3) the framework allows systematic exploration of the visualization techniques design space.

III-3.1 Completeness

Completeness refers to whether the framework is capable of expressing all visualization techniques. At the end of the previous chapter we showed that our framework can express many current visualization techniques. The framework, however, is not complete, and can never be so because as new visualization methods and metaphors are created the framework would need to grow to include these new techniques. It is thus important for the framework to be flexible and easily extensible.

Our framework supports both flexibility and extensibility because it splits the design into two different levels (functional and instantiation). We anticipate that framework extensions will commonly occur only at the instantiation design level because at that level, we are more concerned with input-device specifics and visualization function inputs and outputs. In contrast, the functional level deals with abstract function classes and composition operators, which tend to be less volatile. Changes made to the instantiation design level would generally not affect the functional design level, so framework alterations should be fairly localized. Secondly, the framework is based on a compositional language that allows us to

generate a wide space of designs with relatively few primitives. Future extensions should be able to capitalize on this compositional language and leverage off of pre-existing object definition and transformation functions so that only a few primitives need to be added to increase the expressive capability of the framework significantly. Finally, one of the design decisions was to use object definition and transformation functions of lower granularity (i.e. just simple mathematical functions). This makes it easier to reuse and compose these functions, even with new methods and metaphors. This lower granularity level comes at the price of more specification; however we believe that much of the specification can be automated. In addition partial specifications can always be saved and reused so that we only need to declare them once.

III-3.2 Coverage

Our interactive framework can be applied to a wide range of traditional visualizations (e.g. *charts*, *maps*, *tables*) and direct manipulation input-devices and widgets (e.g. *mouse clicks*, *keyboard presses*, *menus*, *radio-buttons*, etc). The examples in this chapter and the previous chapter show the use of a fairly wide range of visualization types including *maps* (Figure III-11), *charts* (Figure III-2), *tables* as well as input-devices including *sliders* (Figure III-11), *drag and drop*, and *menus* (Figure III-12). This is achieved by building our language based on previous work that have characterized a wide range of data [*Mackinlay, 1986a, 1986b; Roth, 1990*], visualization elements [*Mackinlay, 1986a, 1986b; Roth, 1994; Chuah 1995*], and input-devices [*Card, 1990*]. We do not deal with more complex input-devices, such as two-handed input-devices and speech; however such techniques can be relatively easily integrated into the framework by specifying them according to the three input-device levels described in section III-1.4. Framework generality provides flexibility in design, however, it also raises a big concern, namely how to pick the "best" visualization objects or input-devices from the wide range of choices available. In order to make this decision we must carefully consider our data, our media, and our task [*Bertin, 1983; Tufte, 1983; Mackinlay, 1986a, 1986b; Casner, 1991*]. Choosing the "best" visualization design for our task and data is the topic of discussion in the next chapter.

III-3.3 Practicality

Our framework is practical for the following three reasons.

III-3.3.1 Reduces Cost of Task Tailoring

Our framework provides designer with an easy means of combining common transformation functions to form visualization techniques so that they can plug and play with different visualization effects to suit to suit their design goals. The example shown in section III-2 illustrates this by showing how the dynamic query slider technique can be easily expanded in several simple steps to solve some of the limitations found in the original technique.

III-3.3.2 Provides a New Design Methodology

Our design methodology is based on two different levels of abstraction. The functional level of abstraction (outlined in the previous chapter) is concerned with the goals of a technique while the instantiation level of abstraction (outlined in this chapter) is concerned with the form or look of the technique. By dividing the design process into these two levels, designers can concentrate on functionality first without having to worry about specifics like the *color* to use, whether to use a combined input-device, or whether to use *size* instead of *shape*. At the functional level designers can focus on issues such as whether the functions chosen are capable of solving the task(s) well, whether the functions combine well together, and whether sufficient feedback is provided. Also by concentrating on functionality, designers may notice similarities among techniques that they previously considered to be quite different due to superficial differences. This will hopefully encourage more functional reuse among different techniques. This two-tier design methodology allows designers to focus on different aspects of the design process without complications from other unrelated parts. By doing so we are ensuring that their choice of function is more driven by task concerns rather than by media and device restrictions, which should be dealt with separately.

III-3.3.3 Allows Systematic Exploration of the Visualization Techniques Design Space

The visualization techniques design space can be explored based on the two design levels outlined above: functional and instantiation. Exploration at the functional level involves developing new object definition and transformation functions, as well as combining existing functions in new ways to derive new behaviors. We did this in the previous chapter, which showed how techniques with different functionalities can be combined. Exploration at the instantiation level, on the other hand, involves picking which input-devices, graphical properties (*color, shape, position*), data attributes, or graphical elements to use for a technique. In section III-2 we showed some example instantiation specifications and how these specifications may be varied to produce interesting design alternatives. Note that design variations at the instantiation level do not change the functional characteristics of a technique. The functional space for visualization techniques is bound by the object definition, transformation, and composition classes available. The instantiation space for a given technique is bound by the original functional design of the technique and by the five classes of design changes in the instantiation augmentation process.

III-4 Conclusion

In this chapter we describe the instantiation design of visualization techniques. We show how instantiation details can be added onto functional visualization technique specifications so that they may be rendered as an active visualization interface. Just like the functional level, the instantiation level may be explored in a systematic fashion. Through a series of examples in sections III-2 we show some design variations that may be derived from existing visualization techniques.

We also evaluated the entire framework based on three criteria: completeness, coverage and practicality. In terms of completeness the framework is able to express many current techniques but because of its nature can never be fully complete. It is however easily extensible. In terms of coverage the framework allows various techniques to be integrated with a wide range of visualizations and physical and virtual input-devices. Finally, we argued for the practicality of the framework by showing that it a) allowed easy task tailoring b) provided a new design methodology based on two levels (functional and instantiation) and c) allowed the systematic exploration of the visualization techniques design space. In appendix B-6 we discuss more advanced visualization technique issues that occur when we integrate multiple visualization techniques within a common workspace.

Our analysis of the visualization techniques design space show us that there are many visualization technique alternatives for achieving a single data analysis task or problem. Because of the enormous number of design alternatives, it can sometimes be difficult and time-consuming to test out all the design variations. An automatic design system would help designers create and generate their prototypes more quickly and easily. Such a system however requires the framework, which we have laid out in this chapter and the previous one. Our visualization technique framework provides an automatic design system with a language for describing visualization techniques and a systematic methodology for creating and exploring the visualization techniques design space.

This chapter and the previous chapter described a framework of visualization techniques that had data, mapping, graphical, and rendering transforms. We did not, however, explore the effectiveness of these transform functions. Earlier work on automatic visualization design considered effectiveness criteria for mapping transforms based on data and task requirements [*Mackinlay, 1986a, 1986b; Casner, 1991*]. In the next chapter, we consider effectiveness criteria for making combined decisions about data transforms, and mapping transforms. Specifically, we consider when it might be more useful to perform a task or subtask perceptually by mapping it to graphics, and when it might be more advantageous to let the system internally compute the task through data transforms and only visualize the pre-computed results. In appendix F, we consider the role of graphical and rendering transforms in improving the readability of a visualization. Specifically, we consider readability issues such as occlusion, display density, data dwarfing, and information presence, and how these issues affect the usability of a visualization interface.

Chapter IV: Design Heuristics

Data Computation vs. Perceptual Mapping

The goal of this thesis is to enhance the breadth and quality of designs generated by an automatic visualization system by adding data processing operations to the design process. The consideration of data transforms extends previous automatic systems, which only considered mapping transforms (i.e. mapping data to graphics). In the previous chapters we laid out a framework which divides the visualization design process into four primary steps: data transforms, mapping transforms, graphical transforms, and rendering transforms. In this chapter, we discuss integration of data transforms together with mapping transforms, the issues that arise, and the design improvements that may be realized with this expansion. In appendix F, we speculate about the integration of graphical and rendering transforms into the automatic design process.

We begin this chapter with an example that shows where previous work leaves off, and how consideration of data transforms will improve the designs that can be produced.

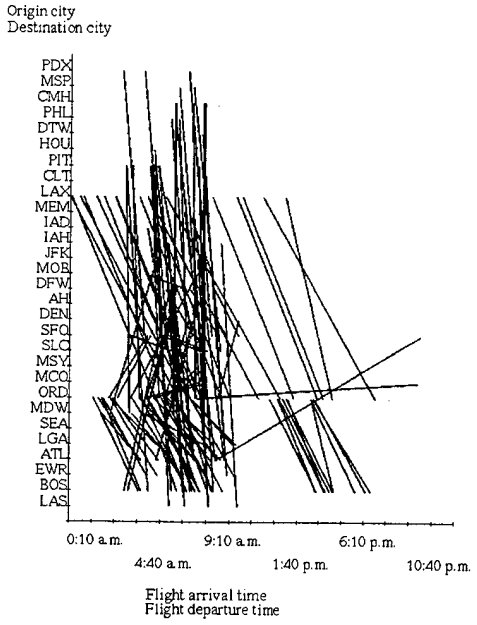
IV-1 An Airline-Scheduling Example Illustrating the Use of Data Transforms

This airline-scheduling example was used by Casner [*Casner, 1991*], to illustrate the importance of considering a user's task in mapping data to graphics (i.e. mapping transforms). We show that this "airline reservations" task can be better supported if we consider ways to transform or reorganize the data in addition to mapping it to graphics. The verbal description of the task that the visualizations must support is:

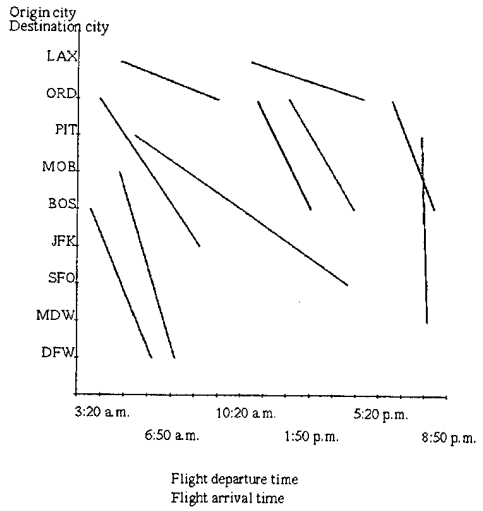
> *Given an origin and a destination city, the user "attempts to locate the two flights arriving in and departing from a layover city that offer the minimum amount of `down time' between the flight times and the beginning and ending time of a scheduled meeting (in the layover city)".*

Task IV-1: Airline-scheduling task. The user is trying to find flights to enable a meeting to be held in a layover airport en-route to a destination and to minimize time spent at the layover airport before and after the meeting.

In the following example designs, suppose that the origin and destination cities are *Los Angeles (LAX)* and *Boston (BOS)* and that the layover city is *Chicago (ORD)*. Further, suppose that the meeting time is from 2 p.m. to 4 p.m. Casner showed that this task can be achieved perceptually with the graphic design in Figure IV-1. In Figure IV-1, the *origin* and *destination* cities are encoded on the *y-axis* and the *departure* and *arrival* times are encoded on the *x-axis*. Each flight is represented by a line where the left-point of each line encodes the *origin city* and the *departure time* of a flight and the right-point of each line encodes the *destination city* and *arrival time* of a flight. For example in Figure IV-1b, the task solution is a flight leaving *Los Angeles* (LAX) at 5:30 a.m. noon and arriving at *Chicago* (ORD) at 10:20 a.m. and another leaving *Chicago* (ORD) at 7:00 p.m. and arriving at *Boston* (BOS) at 9:10 p.m.



(a) Full data set
This visualization shows all the elements in the data set (i.e. all 135 flights).



(b) Truncated data set. This example visualization shows the ideal case where there is little occlusion among the different flight lines. This data set was chosen so that it contains some flights that fulfill the task constraints as well as some other random flights that do not occlude one another.

Figure IV-1: Solving the airline-scheduling task fully perceptually (Casner's solution). Each line represents a flight with origin and destination city mapped onto the *y-axis* and arrival and departure time mapped onto the *x-axis*. This is the best design that gets generated when ONLY mapping operations are considered by the automatic system. I.e. this is the best possible design from current state of the art systems.

In Figure IV-2, we present Casner's analysis of the perceptual procedure a user must perform using the visualization in Figure IV-1 to achieve the airline-scheduling task (Task IV-1). Subsequently we show in Figure IV-3 how transforming the data makes this procedure simpler.

Subtask-1 :*Find all the origin flights that fulfill the first leg of the flight schedule*

Visually search for all flights whose origin is *Los Angeles (LAX)* and whose destination is *Chicago (ORD)*, the *layover-city*.

Search for *LAX* on the *y-axis* and then look over to the right for all flights that start from this *origin-city*.

For each of these flights, we find the end-point of the flight line and determine whether it goes to *ORD*, the *layover-city*.

For all flights that fulfill the origin (LAX) and destination (ORD) city constraints, check if they meet the meeting time constraints as well (arrives at ORD before 2 p.m.).

Look down on the *x-axis* to determine the *time of arrival* in *ORD*. If the time is after the scheduled meeting time, we discard the current flight as a possible candidate and continue looking for other relevant flights.

If the arrival time is before the scheduled meeting, we determine whether it has the smallest prior meeting downtime. If so we note the flight as the current most promising candidate and continue the process for all other flights.

Subtask-2 :*Find all the destination flights that fulfill the second leg of the flight schedule*

Find the earliest flight after the meeting using an analogous procedure to *subtask-1*.

At the end of *subtask-1* and *subtask-2* we would have determined the flights with the smallest downtimes before and after the meeting. To get the *total down time* we merely add the two downtimes

Figure IV-2: Casner's analysis of the perceptual procedure a user must perform with a visualization to achieve the airline-scheduling task (Task IV-1).

The pure perceptual procedure for the airline-scheduling task (Task IV-1), while relatively complex, is still more effective compared to a strictly *cognitive* procedure (i.e. looking at a spreadsheet table that contains the raw data). We performed a GOMS analysis for a tabular presentation of the data for the airline-scheduling task and estimated it to take approximately 4 minutes for task completion. In contrast, the perceptual solution represented in Figure IV-1only took 30 seconds. The GOMS analysis tables for both the cognitive and perceptual designs are presented in appendix C-1. However, we should point out that this analysis assumes that all the data fits within a single CRT screen, and there is no occlusion in the designs.

As can be seen from Figure IV-1a, when the data set size grows, the perceptual design quickly becomes unusable without interactive navigation of the display. For the cognitive solution, larger data sets would not fit within a single CRT screen thus interactive scrolling is required. Such navigation operations will add to the overall task time of both designs.

Figure IV-3 shows an alternative design for solving the same airline-scheduling task using the same data set as Figure IV-1b. The left chart shows all the flights that fulfill constraints for the first leg of the journey (*LAX* to *ORD* with *arrival time* before 2 p.m.) while the right chart shows all the flights that fulfill constraints for the second leg of the journey (*ORD* to *BOS* with *departure time* after 4 p.m.). The bar lengths in the left chart encode the computed total downtime before the meeting and the bar lengths in the right chart encode the computed total downtime after the meeting.

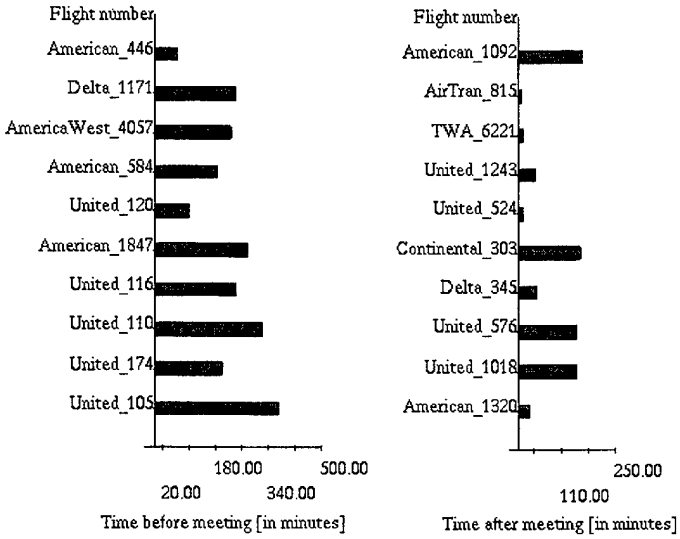


Figure IV-3: Our hybrid data transform and mapping transform design for solving airline-scheduling task. Here only the flights that fulfill the *city* and *meeting time* constraints are shown. Computation of the total downtime for the best flights is left to the user. *Time_before_meeting* is mapped to the *x-axis* of the left chart and *time_after_meeting* is mapped to the *y-axis* of the right chart. To perceptually compute the total downtime users add the shortest bar length in the left chart with the shortest bar length in the right chart.

In Figure IV-3, we are able to significantly simplify the perceptual complexity of the earlier designs as well as reduce visual clutter with **data transform techniques**. These techniques allow the automatic system to summarize the task results and filter out irrelevant flights thereby significantly improving the readability of the representation compared to Figure IV-1a. In addition, it also simplifies the perceptual procedure for solving the task because in this design users need not visually search for flights that fulfill the *city* and *time* constraints. Instead, all this information has been pre-calculated by the system with data transform techniques. To solve the airline-scheduling task (Task IV-1) using Figure IV-3 we only need to pick the shortest bar in the first chart (i.e. *American_446* which has the least downtime before our meeting)

and the shortest bar in the second chart (i.e. *AirTran_815* which has the least downtime after our meeting). The total downtime can be estimated by perceptually adding the lengths of both these bars. The GOMS analysis for Figure IV-3 showed an estimated task time of only 3 seconds.

Figure IV-4 shows the GOMS time estimates for solving the airline-scheduling task using a purely cognitive procedure, a purely perceptual procedure (i.e. only mapping transforms, Figure IV-1), and a perceptual procedure (mapping transforms) combined with data transform techniques (Figure IV-3). We can clearly see that using both mapping and data transform techniques together is significantly more effective than using only mapping transform techniques which in turn is more effective than using only cognitive operators.

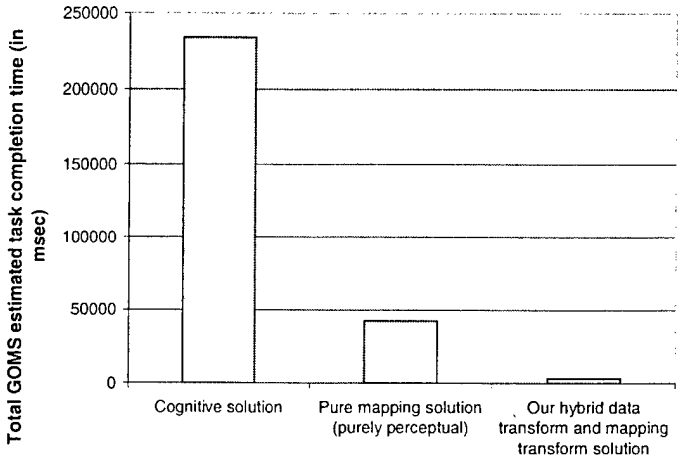


Figure IV-4: GOMS estimated total time for solving the airline-scheduling task using a pure cognitive, pure mapping, and a hybrid data + mapping design.

Earlier work on automatic visualization design [*Mackinlay, 1986a, 1986b; Casner, 1991, Roth, 1994*] centered purely on using mapping transform techniques (pure perceptual operators), preferring to address those tasks that cannot be easily accomplished through statistical computation (data transform techniques). This sentiment is well expressed by Tufte: "Why waste the power of data graphics on simple linear changes which can usually be better summarized in one or two numbers? Instead, graphics should be reserved for the richer, more complex, more difficult statistical material." Thus, previous work on automatic presentation systems have assumed that statistical processing has already occurred before the design process. These systems have instead focussed on developing design heuristics for making data-to-graphical mapping decisions based on both tasks and data. However, work on task analyses [*Springmeyer, 1992*] indicates quantitative processing (data transform techniques) is an integral part of graphic design. As was expressed by Springmeyer, "These results show that analyzing scientific data is a much more quantitative and active process than the passive viewing of images." Based on her analyses, Springmeyer was convinced that a large shortcoming of current visualization systems was their lack of integration with quantitative operations (i.e. data transforms). We believe that in order to design more effective

visualizations for analyzing data, we must make data transform decisions together with mapping transform decisions. The brief GOMS comparison shown in Figure IV-4 supports this belief.

It is however erroneous to assume that data transforms will always be more effective than mapping transforms. Sometimes, over-computing a task creates more graphics and more work for the user (appendix C, Figure C-3). Other times, full pre-computation is just not possible. Commonly the most appropriate design for a task will consist of a blend of data and mapping transforms, as we will show in the next example. The most effective blend of data transforms and mapping transforms for a task sequence is dependent upon many factors including the task (e.g. whether the task requires simple or complex computation, whether the operation must be repeated many times, whether we know for sure what the task parameters are), the preferences of the user (e.g. whether they are comfortable with using input devices, whether they are familiar with particular input devices), and the availability of display and input resources (e.g. whether the elements will fit within the output media, whether physical devices are available for input). To make intelligent design decisions about how data computation and mapping techniques should be combined, we need to include data processing decisions as part of the automatic design process and not merely pre-compute the data beforehand. In the next example, we illustrate the weakness of over-computing the airline-scheduling task (Task IV-1).

One possible solution for Task IV-1 is to calculate the entire task with data transform techniques. In this case, the system would only present users with the two flights that produce the minimum *total downtime* (i.e *American_446* and *AirTran_815*). Having the system calculate the entire task with data computations, however, is only appropriate if we can fully and accurately define **all** our data analysis goals (e.g. meeting time constraint: >= 2 p.m. and <= 4 p.m.; origin = *LosAngeles;* layover = *Chicago;* and destination = *Boston*). Suppose that in addition to *total downtime* we were also concerned with *total cost* and *flight duration* (i.e. we want to choose flights with "generally low" *total downtime, total cost,* and *flight duration*). In this case, it is not possible for the system to calculate the entire task with data computation and only present users with one flight pair because we do not know what constitutes an acceptable balance between "low" *total downtime*, "low" *total cost* and "low" *flight duration*. The best balance between these three attributes can only be arrived at during the analysis process, after we have determined the number of flights that fulfill our *city* and *time* constraints and the data distributions of the acceptable flights with respect to our three attributes.

One way to solve this task is to use a design similar to the one in Figure IV-3 but augmented with *total cost* and *total duration* data (Figure IV-5). The left chart in Figure IV-5 represents flights that fulfill the first leg of our schedule and the right chart represents flights that fulfill the second leg of our schedule. In each chart, the labeled marks represent different flights. The *flight duration* is pre-computed, and encoded on the *y-axis*, *total downtime* is pre-computed and encoded on the *x-axis*, *total cost* is pre-

computed and encoded with saturation, and *flight name* is encoded with *labels*. To find the most relevant flights, we look for marks on the left-bottom corner of each chart (low *downtime* and low *flight duration*) with low saturation values (low *total cost*). In the left chart (flight before meeting) the choice is clear. The best flight is *American_446* that has the lowest *flight duration* and *downtime_before_meeting*. Its cost (*saturation*) is also comparable with the other flights. In the right chart, some trade-offs must be made between *AirTran_815* that has the lowest *downtime_after_meeting* and *cost* and *United_576* that has the lowest *flight duration*. *AirTran_815* seems to be the better choice from Figure IV-5 because it has very low *downtime_after_meeting* and *cost*, as well as a *flight duration* that is not overly large whereas *United_576* has a very large *downtime_after_meeting*. Thus, this airline-scheduling task cannot be solved with a pure computation because the tradeoffs among the task attributes (*downtime*, *flight duration*, and *cost*) cannot be captured in a simple function and needs to be considered by the user. At the same time however, using a pure mapping design is also ineffective because of its high task complexity (in terms of both search and computation) as well as the many different data attributes that it combines. Thus, a hybrid data and mapping design (as in Figure IV-5) is the most appropriate here.

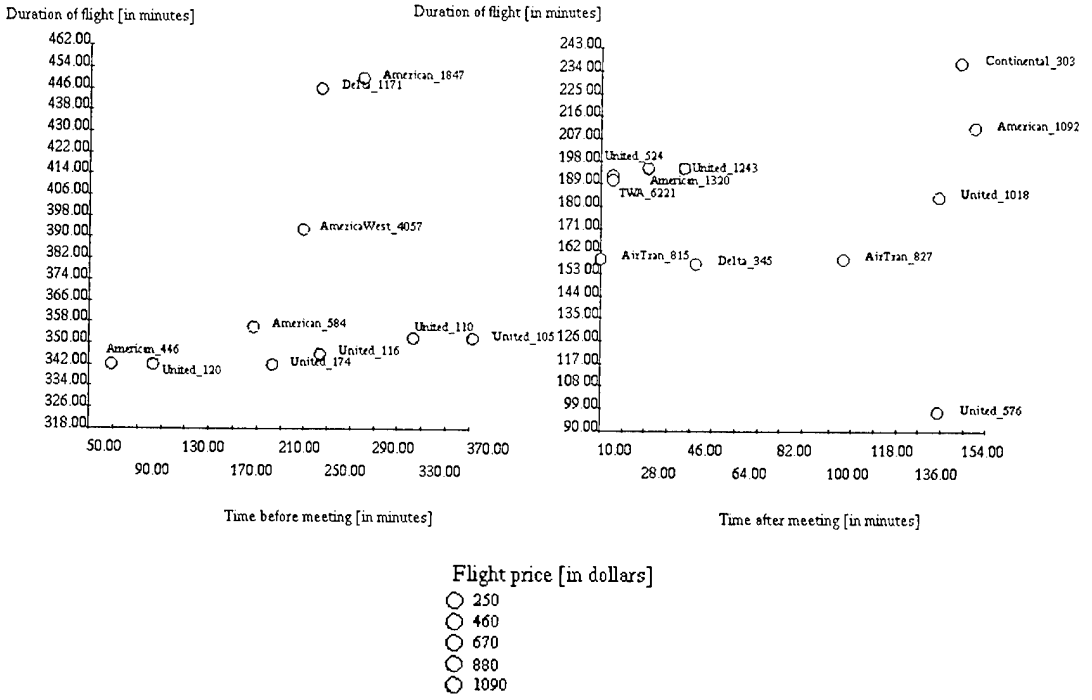


Figure IV-5: Visualization for finding flights with low total-downtime, low total-cost, and low duration. Because there are trade-offs that must be made among the three attributes, this task is best performed through perceptual perusal.

In appendix C-2 we present several other design alternatives for the airline scheduling task that have different blends of data and mapping transforms and discuss their strengths and weaknesses. The examples in this section and in appendix C-2 show us that there are many ways with which we can use data and mapping transforms to solve user goals. To make intelligent design decisions about how data computation

and mapping techniques should be combined, we present a set of design dimensions for gauging the goodness of different visualizations (section IV-2). Based on these design dimensions, we develop higher level design rules for deciding when to use data transform techniques and when to use mapping techniques to solve tasks (section IV-3). In appendix C-4 and C-5 we introduce some task examples and show how our prototype automatic visualization designer addresses these tasks based our design dimensions and design rules.

IV-2 Visualization Design Dimensions

The airline-scheduling example presented in the previous section shows many different ways in which data and mapping techniques may be combined. To decide on which combination is most appropriate for a given task or set of tasks we need some standards of evaluation for the different designs (i.e. visualization design dimensions). In this section we present a set of design dimensions upon which to evaluate visualizations. Our dimensions are based on the interaction framework model presented by Abowd and Beale [*Abowd, 1991*] (Figure IV-6).

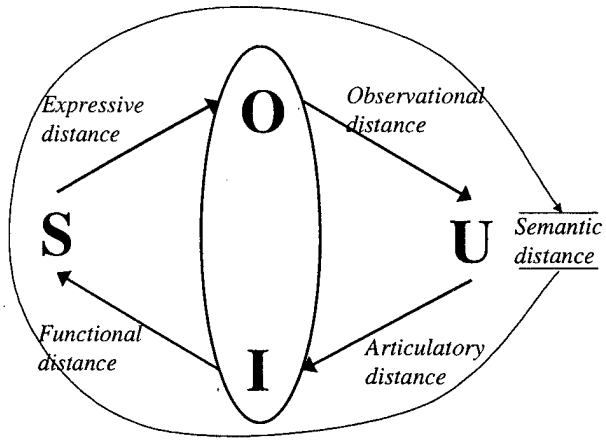


Figure IV-6: Interaction Framework model presented by Abowd and Beale [*Abowd, 1991*]. This framework is used to measure the effectiveness of various visualization interfaces in this work.

In this framework, there are four components: the user (*U*), the system (*S*), input (*I*), output (*O*); as well as four translations between these components: *articulation, performance, presentation, observation.* Users through *articulation* generate inputs for a system detailing the requirements of their current tasks. The system *performs* a set of function operations on these inputs and generates a set of outputs. These outputs *present* a possible solution of the users' input queries. Users must finally *observe* and interpret these system outputs, updating their task model as necessary. This cycle is iterated over as many times as necessary until all task objectives have been satisfactorily met. Each translation step can be assessed for its effectiveness with respect to the overall interaction. Effectiveness of the four translation steps is measured by their *articulatory distance, functional distance, expressive distance,* and *observational distance* respectively (Figure IV-6). The summation of these four distances measures the effectiveness of the overall

interaction or its "*semantic distance*". *Semantic distance* refers to the degree which user goals are fulfilled by the interaction. A large semantic distance means that the goals are not achieved well and a small semantic distance means that the goals have been satisfied acceptably.

Articulatory distance measures the ease with which users can specify their desires to the system. For visualization systems, articulatory distance measures the amount of input device manipulation required from users. A visualization technique that requires a great deal of user input has high articulatory distance and vice versa. *Functional distance* refers to whether the system possesses software functions or procedures capable of achieving user tasks. In our case, functional distance refers to whether the object definition functions (enumeration and functional description) and the transformation functions (data, mapping, graphical, and rendering) presented in the previous two chapters are sufficient to support basic data exploration tasks. *Expressive distance* determines whether sufficient feedback or information is provided to users to solve the input tasks. "*Sufficient feedback*" may mean whether sufficient data concepts and relationships are provided to solve the input tasks, whether the presentation reflects all facts contained within the data set, whether false information is introduced, and/or whether all information contained within the visualization is displayed at all times[1]. Finally, *observational distance* refers to the ease with which a user can interpret system feedback. Specifically, *observational* distance measures the effectiveness of the visual objects, visual properties, and visual compositions used to fulfill the input analysis tasks[2].

Our design dimensions measure either articulatory, expressive or observational distance. As for functional distance, we have supplied our system with all the necessary object definition (e.g. *enumeration, functional description, set operations*) and transformation functions (e.g. *addition, subtraction, assignment, grouping, mapping data to graphics*) needed to perform the basic set of data exploration tasks used in previous automatic system research and which we find interesting on our own work. Thus, the functional distance measure is not pertinent in our case. More generally, completeness of our object definition and transformation functions with respect to existing visualization systems was discussed in the previous two chapters. Completeness of our system in terms of task coverage is described in appendix C-3. We will now describe the various dimensions that may be used to estimate articulatory, expressive or observational distances as well as how these distances may be used to gauge the effectiveness of a design that favors data transform techniques versus one that favors mapping techniques.

---

1 The expressive distance described here is an expansion of Mackinlay's expressiveness criteria [*Mackinlay, 1986a, 1986b*].

2 Observational distance corresponds largely to Mackinlay's effectiveness criteria [*Mackinlay, 1986a, 1986b*]. In his dissertation Mackinlay presented a set of effectiveness heuristics that ranked different graphical properties (perceptual operations) based on their perceptual accuracy.

IV-2.1 Articulatory Distance

Articulatory distance increases with the amount of user input required by a visualization interface. When using visualizations to solve data analysis problems, user input is required for two primary purposes: 1) task clarification/alteration, and 2) data navigation as is shown in Figure IV-7.

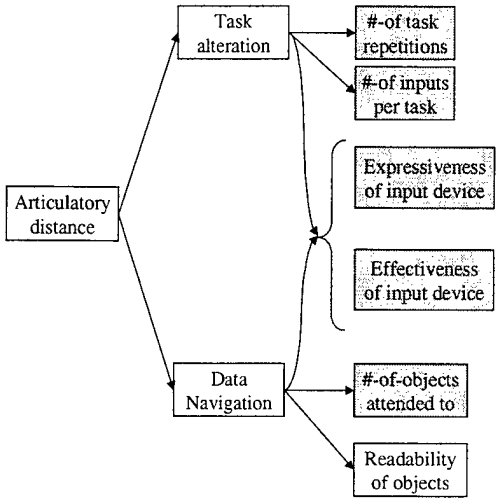


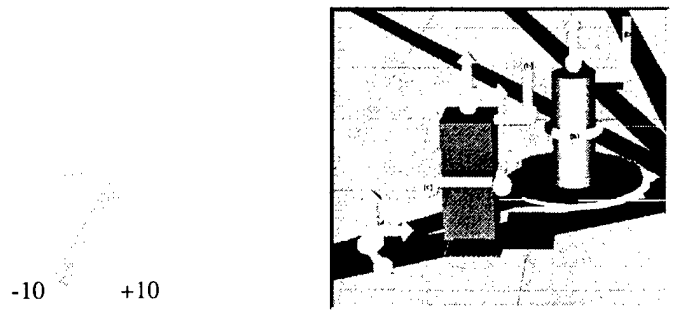
Figure IV-7: Breakdown of articulatory distance. Gray highlighted rectangles indicate the dimensions that are taken into account in our prototype automatic presentation system described in chapter V.

a) *Task alteration/clarification load*: To solve some data analysis tasks a user may need to provide task arguments to the computer system interactively. The amount of articulatory load required depends on the number of times a task needs to be repeated multiplied by the number of inputs per task. When a task is repeated many times, it becomes very important to reduce the articulation load of each iteration, even at the cost of losing flexibility.

b) *Data navigation load*: Data navigation operations are commonly required for larger data sets where there is clutter and occlusion in the graphic design resulting in high observational distance. It is possible to lower observational load by limiting the amount of information that is shown to the user at any one time. The disadvantage, of course, is that users must navigate to different pertinent information slices through input devices. Data navigation depends on the number of objects that must be attended to and the readability of those objects (e.g. whether they are occluded, too small to interpret, or surrounded by high ink density. *Readability* issues are explored in detail in appendix F). The more objects we need to attend to, the greater the likelihood that we must perform more navigation. In addition, the less visible or readable the objects are, the more effort we must expend to get them to a readable state. In our prototype designer, we only estimate data navigation load by the number of objects attended to, leaving the more complex readability issues for future work.

c) *Expressiveness of an input device*: An input device is considered expressive of a particular visualization function if it can be used to generate all the inputs required by that function **and only** those inputs. A *menu* for example, is only expressive of discrete values, i.e. it can only be used to pick from a finite set of set of strings or numbers, but cannot be used to generate continuous input values. A *slider*, on the other hand, can be used to generate both continuous and discrete input arguments. *A text window* is very flexible and can be used to specify any input argument type. However, it is generally not very expressive because it does not indicate to users what the acceptable input arguments are unlike the *slider* and *menu* devices. I.e. users may very easily enter invalid input values when using *text-windows*. Our design system only allows the use of input devices that are expressive of the input data or arguments required. Unlike the previous two dimensions which are quantitative measures, expressiveness of input devices is implemented as a binary measure in our system.

d) *Effectiveness of an input device*: The effectiveness of an input device measures how easily users can manipulate an input device to generate the required task arguments. The effectiveness of an input device is most commonly measured by the amount of motoric energy expended by users for each input entry. Input devices such as buttons require very little motoric load because users only need to move the mouse over the button and click. Menus have a higher motoric load because we need to scroll down a list of choices in addition to choosing an entry with a mouse move and click. A text window has the highest motoric load because we need to move our hands over to the keyboard and type out our entries, which could take multiple keystrokes. Therefore, when the required input arguments can be expressed by a *button* input device (i.e. the input value range must be small and discrete), it should be chosen instead of *menus* and *text windows* because it is the most ***effective*** device.



(a) Dial **(b) SDM handles[*Chuah, 1995b*]**

Figure IV-8: Input devices with different effectiveness properties

Input device effectiveness can also be measured by how easy they are to learn. Some input devices provide good *affordances* (or cues) to users indicating how they may be manipulated. For example, a *dial* or *knob* (Figure IV-8a) is an effective device for producing radial values because it provides good affordances for showing users that it should be rotated. In contrast, the virtual object handles provided in the SDM system (Figure IV-8b) are less effective because it is less clear how they should be

manipulated and what they control. In our system we only account for effectiveness based on ease of use (level of effort) and not learnability. The input devices in our system are ordered in a list based on fewest manipulations to most manipulations. The system then picks the first expressive device in the list (i.e. the first expressive device that is most effective). Details of this process are described in chapter V.

In general, performing a task with data techniques requires more articulatory load because the task must be very explicitly stated (there can be no missing values). When there are unknown task arguments, the data technique designs require that users provide these missing arguments to the system. Thus, articulatory load is high because either we must provide several different initial task specifications, each containing a different task argument alternative or we must provide task clarification/alteration parameters during the data analysis process. The task specificity guideline in section IV-3.6 reflects this property of data techniques.

IV-2.2 Expressive Distance

Expressive distance measures whether sufficient data or information is shown to the user. What actually constitutes "sufficient data" may be interpreted in several ways: a) expressiveness of task, b) expressiveness of data, c) data correctness, and d) data presence, as is shown in Figure IV-9.

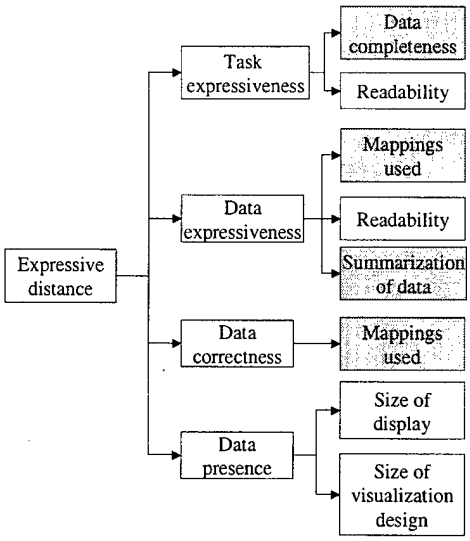


Figure IV-9: Breakdown of expressive distance. Gray highlighted rectangles indicate the dimensions that are taken into account in our prototype automatic presentation system described in chapter V.

a) *Task expressiveness*: In order for a visualization to be expressive of a data analysis task, there must be a sequence of cognitive, perceptual, and motoric actions that users may perform on the visualization design that will result in a solution to their task. These sequence of actions can only be generated if the visualization design contains all the data concepts necessary for solving the task (data completeness) and presents this information in a way that is accessible to users (readability). For example to solve the

desired airline-scheduling task (Task IV-1), the visualization may contain the *origin-city, destination-city, arrival-time* and *departure-time* of all flights as in Figure IV-1. Alternatively, it may include only those flights that fulfill our *city* and *meeting time* constraints as in Figure IV-3. Although these two visualizations contain different data concepts and attributes, both contain enough information for solving the airline-scheduling task. Thus, Figure IV-1 and Figure IV-3 both have data completeness. On the other hand a visualization that only shows the *origin* and *destination* cities of all flights (leaving out their arrival and departure times) is insufficient for picking flights with the minimum *total-downtime* because no time information is provided to users. Such a design, therefore, is not expressive of the airline-scheduling task.

To achieve task expressiveness, a visualization must not only contain all the task information, but this information must be accessible to users. Sometimes, due to problems such as occlusion, dwarfed objects[3], or display density, some of the encoded information may not be visible or readable by users. For example, Figure IV-1b is data complete but not task expressive because some of the encoded information cannot be accessed due to object occlusion. We discuss readability issues in the appendix F as well as outline how they can be addressed using graphical and rendering transforms. In this chapter, we show how some readability problems may be avoided with appropriate combinations of data and mapping techniques as in Figure IV-3.

b) Data expressiveness (Information loss wrt. original data set): A visualization is generated by processing and mapping a set of data concepts and attributes to graphics. We call the set of original data concepts and attributes the *original data set*. This data set is commonly attached to data characterizations that describe the concepts and attributes contained within the set, as well as the relationships among the data [*Mackinlay, 1986a, 1986b; Roth, 1990*]. These data characterizations help us structure the data so that we can generate better design solutions. Generally however, not all of the concepts or data characterizations contained within the original data set must be shown to solve a given set of analysis task(s). That is why data expressiveness is different from task expressiveness.

For example, data transform techniques may cause information from the original data set to be lost through data summarization or culling. In Figure IV-3 much of the flight data from the original data set was filtered out, thus Figure IV-3 is *task expressive* but not *data expressive*. Data technique designs are usually much less data expressive compared to mapping technique designs because they work by simplifying or summarizing data and only showing the results of those simplifications.

c) Correctness of the visualization (Information integrity): The expressive distance of a visualization also depends on its correctness[4]. Certain graphical languages may imply facts about the encoded data values

[3] Problems with scale that prevent some values from being differentiated.

[4] The concept of data correctness was first introduced by Mackinlay [*Mackinlay, 1986a, 1986b*].

that are untrue. For example using *saturation* to represent an unordered attribute (e.g. *house neighborhood*) suggests a perceptual ordering when actually there is none. Figure IV-10 shows a set of houses, represented as marks arranged in a grid representation. The *saturation* of the marks indicates the *house neighborhood* attribute, which is an unordered attribute. However, because *saturation* is an ordered perceptual property, the visualization falsely shows that the *Pt.Breeze* neighborhood (most saturated) is ordered above the *Squirrel Hill* neighborhood (less saturated) which is ordered above the *Shadyside* neighborhood (least saturated).

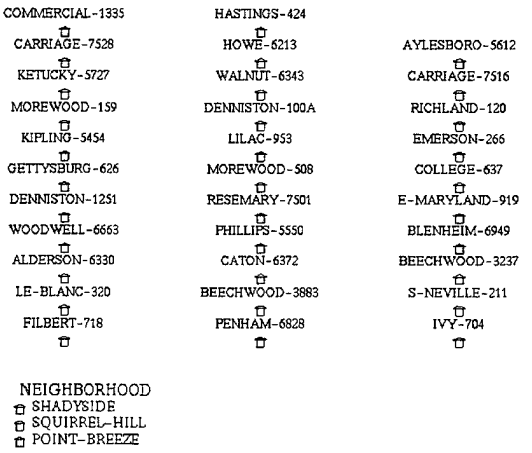


Figure IV-10: Encoding house neighborhood with saturation, This encoding has low data correctness because *saturation* is an ordered graphical property while *neighborhood* is not an ordered data attribute. By using *saturation* to encode *neighborhood* we are falsely implying an ordered set of *neighborhood* values when actually there is none.

d) Data Presence: Sometimes a visualization design is too large to fit within the CRT screen. When this occurs we must divide the visualization into segments and display sub-portions of it to users at different times. The visualization design is therefore only expressive of a piece of information for a limited time (i.e. temporary expressiveness). Data presence measures the ratio between the information shown per instance on the CRT screen with respect to the information within the entire visualization. Generally, a visualization with low data presence is also less expressive because only a small part of the total information can be seen at any one time. There are two ways to measure data presence: by calculating the ratio between average number of objects shown per instance and the total number of objects, or by calculating the ratio between visualization space per instance with respect to the entire visualization area. The lower the object or spatial ratios, the less data is shown and the greater the probability that users may miss some of the information and misinterpret the data contained within the visualization. When data presence is less than 1 (i.e. some information is hidden) users may find it necessary to store some information in short term memory to maintain context between the different information slices. This increases the cognitive load (observational distance) placed upon users.

Expressiveness criteria for visualizations was first introduced by Mackinlay [*Mackinlay, 1986a, 1986b*]. Mackinlay defined expressiveness as follows:

> *"A set of facts is expressible in a language if the language contains a sentence that encodes every fact in the set and does not encode any additional facts".*

This definition covers *data expressiveness (b)* and *visualization correctness (c)*. The expressive distance dimensions presented in this section expand on Mackinlay's expressiveness criteria to include two other criteria: *task expressiveness,* and *data presence*. In our system, we account for all the expressiveness dimensions in Figure IV-9 except for *data presence*. Task expressiveness and correctness are implemented in our system as binary constraints (i.e. all designs generated by our system are task expressive and correct) and data expressiveness is implemented as a quantitative constraint.

IV-2.3 Observational Distance

Observational distance consists of cognitive and perceptual loads placed upon users when interpreting results from the visualization system. Perceptual load is determined by the number of perceptual operations that must be performed, and the difficulty of those perceptual operations. Similarly, cognitive load is determined by the number of cognitive operations that must be performed, and the difficulty of those cognitive operations, as is shown in Figure IV-11.

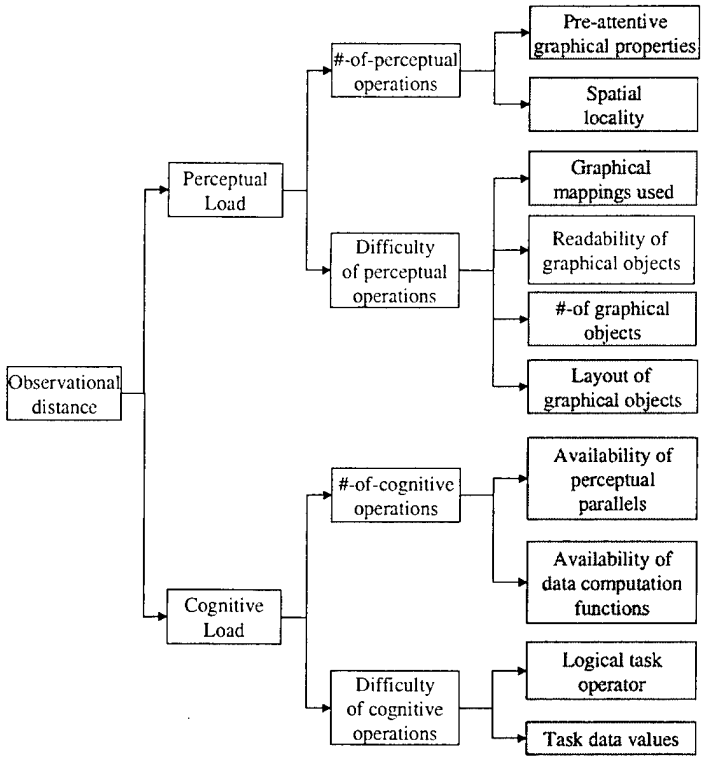


Figure IV-11: Breakdown of observational distance. Gray highlighted rectangles indicate the dimensions that are taken into account in our prototype automatic presentation system described in chapter V.

a) *Number of perceptual operations*: The number of perceptual operations required depends primarily on the effectiveness of the graphical mappings used to represent the data and the task. Certain graphical mappings (e.g. color) enable pre-attentive perception, which allows us to see certain common facts about a set of objects simultaneously (i.e. we need not attend to each object separately). Consequently, the number of perceptual operations required is significantly reduced. Pre-attentive vision is also very useful for quickly filtering out unrelated objects so that we only attend to the ones that are pertinent to our task. For example to find the first flight in the airline-scheduling task presented at the start of this chapter (Task IV-1) the user only needs to consider those flights whose arrival time is before the meeting (i.e. in Figure IV-1 we only consider lines which end before a certain distance to the right). All other flights may be perceptually filtered out. The number of perceptual operations can also be lowered by reducing the number of eye-movements that must be performed. This can be achieved by placing objects with related information together so we do not need to associate objects that are separated over large spatial distances. In our system we give preference to designs that effectively uses pre-attentive graphical properties and have good spatial locality (high level of graphical element integration).

b) *Difficulty of perceptual operations*: The difficulty of perceptual operations depends on the graphical representations and properties used to show the data as well as on the readability of those graphical representations. Different graphical mappings can result in simpler or more complex perceptual operations. For example, to solve an addition task, it is expeditious to map the task values to *stacked-bars* because judging *length* or *position* (e.g. looking up a bar *length*) is easier than computing sums of *size* or *length* (e.g. adding the *length* of two bars). Readability issues can also affect the difficulty of perceptual operations as is discussed in appendix F.

The perceptual complexity of a visualization is also dependent on the overall layout of the design and the number of graphical elements within it. To keep complexity low, we must ensure that the graphical elements and input device controls within the visualization are well integrated. In addition, we also want to ensure that there are not overly many graphical elements or controls, so that the visual interface does not appear too cluttered or confusing to the user.

c) *Difficulty of cognitive operations*: The difficulty of cognitive operations depends on the task. Tasks that require simpler mathematical operations, e.g. *addition* or *subtraction* or tasks that only require simple value comparisons can be solved with lower load cognitive operators. Other tasks such as computing *ratios*, *integrals* and *derivatives*, finding data trends, or identifying data relationships are harder to perform cognitively. The difficulty of cognitive operators may also depend on the data values involved in the operation. For example, it is more difficult to perform computation on numbers that have a higher number of significant figures, e.g. (200 + 300) vs. (273 + 329).

d) *Number of cognitive operations*: The number of cognitive operations required depends on how easily they can be offloaded onto our perceptual system. We usually want to keep the number of cognitive operations to a minimum, because they are usually much harder to perform and more taxing on users compared to perceptual operations. This is easy to accomplish when there are graphical objects capable of expressing the desired task data and relationships. The *addition* task, for example, has a close perceptual parallel - namely *stacked bars*. Thus, the cognitive load can be easily transferred onto the perceptual system. However, this is less true for more abstract computations like *log* and *exponent* which does not have a close perceptual parallel. However, because we consider data transform techniques in our automatic design process, we can offload these more complex tasks onto the computer system through data pre-processing operations. The advantage of data computation is that they offload the entire cognitive operation onto the computer system and only incur a small perceptual load from the user for interpreting the results. This is especially useful for cognitive tasks that cannot be easily mapped to perceptual operations.

IV-3 Data Techniques vs. Mapping Techniques Design Guidelines

The design dimensions given in the previous section provide useful guides for directing an automatic design system to more promising paths in the design space. However, the design dimensions alone are insufficient because some design dimensions are difficult to calculate or measure without additional perceptual and design knowledge. For example using color often reduces the number of perceptual operators and thus the observational distance of a design because it allows for pre-attentive perception. This information however cannot be deduced from the design dimensions alone. The fact that color allows for pre-attentive perception must be encoded into the designer as well. Thus in addition to the design dimensions, we present a set of higher level knowledge guidelines that capture how particular design decisions may affect the "goodness" dimensions of a design.

Previous work on automatic visualization design developed a set of *mapping technique guidelines*. Mapping technique guidelines capture knowledge on how data attributes should be mapped to graphical properties and objects. These guidelines describe the effectiveness of graphical properties for showing different types of data attributes. This could be based on whether the graphical property reduces the number of perceptual operations (e.g. because of pre-attentive perception) or the complexity of the perceptual operations. Mapping heuristics may also include structural heuristics that describe how data attributes should be mapped to objects and how new objects should be combined with existing ones. For example, integration of graphical properties within the same object or cluster of objects is preferred over spreading the properties over multiple regions, because integration reduces the number of eye movements that are required. Chapter V contains more details on how these heuristics can be translated into concrete

constraints and design costs within an automatic design system. For more details refer to previous work on automatic visualization design [*Mackinlay, 1986a, 1986b; Roth, 1990*].

In this section, we focus on defining a set of design guidelines for making decisions between using data transform techniques versus mapping transform techniques to solve tasks. These guidelines were derived using the three distances (*articulatory, expressive,* and *observational*) described in the previous section. Each guideline helps reduce the semantic distance of a task by reducing one or more of these translation distances. Note that these guidelines are not meant to be a complete list of design principles, nor do we claim that they are applicable for all task situations. We do believe, however, that they are a reasonable set of rules for the data analysis tasks that we consider in this thesis. It is important to recognize that these design rules are not meant to replace the expertise of a graphic designer or an information specialist. However, by integrating such design knowledge into an automatic system we hope to enhance the computer system's ability to convey more complex information as well as reduce the more mundane and straight-forward design work that needs to be performed and free visualization designers to explore a much wider range of design alternatives.

IV-3.1 Accuracy

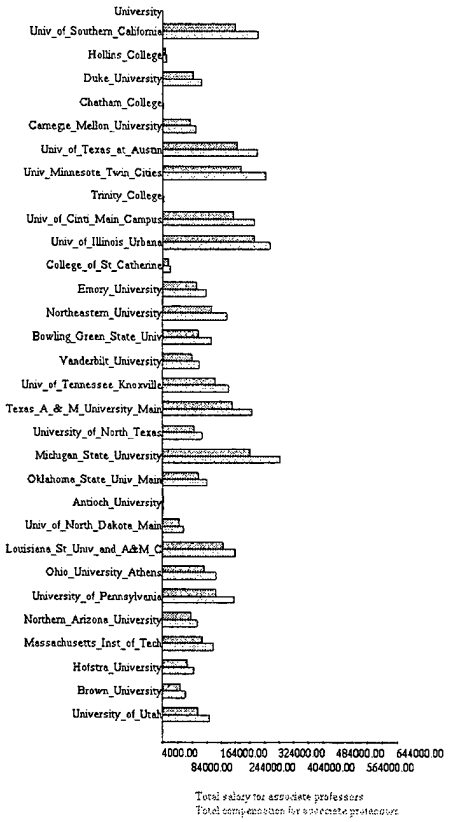


Figure IV-12: Graphic for determining the total benefits for associate professors by getting the difference between total compensation (blue bar) and total salary (red bar)

Different tasks require different degrees of accuracy. When tasks require a high degree of accuracy, there is a preference for using data transform techniques. This is because to get the same level of accuracy through mapping techniques (i.e. perceptual processing), we would need to encode the data as text and then offload the computation process to the user's cognitive system. For example, suppose we were considering a group of universities for possible associate-professorship positions and we want to determine the total benefits given out to associate professors in those universities. In order to get the total benefit values we must determine the difference between the *total_compensation_associate_professor* and *total_salary_associate_professor* attributes in our data set. One possible mapping design to achieve this task is to represent the *compensation* and *salary* values with two sets of bar lengths as in Figure IV-12. However, if we wanted to determine the *total benefit* figures with high accuracy Figure IV-12 is inappropriate because bar lengths can only show the results with a limited amount of precision.

University	Total benefits for associate professors
Univ_of_Southern_California	62658.00
Hollins_College	3078.00
Duke_University	21456.00
Chatham_College	1272.00
Carnegie_Mellon_University	17250.00
Univ_of_Texas_at_Austin	53997.00
Univ_Minnesota_Twin_Cities	67860.00
Trinity_College	876.00
Univ_of_Cinti_Main_Campus	56990.00
Univ_of_Illinois_Urbana	42120.00
College_of_St_Catherine	4187.00
Emory_University	24603.00
Northeastern_University	42640.00
Bowling_Green_State_Univ	34556.00
Vanderbilt_University	20640.00
Univ_of_Tennessee_Knoxville	37700.00
Texas_A_&_M_University_Main	53848.00
University_of_North_Texas	21721.00
Michigan_State_University	83980.00
Oklahoma_State_Univ_Main	24700.00
Antioch_University	1635.00
Univ_of_North_Dakota_Main	12152.00
Louisiana_St_Univ_and_A&M_C	34830.00
Ohio_University_Athens	31812.00
University_of_Pennsylvania	50160.00
Northern_Arizona_University	18432.00
Massachusetts_Inst_of_Tech	29913.00
Hofstra_University	18602.00
Brown_University	14400.00
University_of_Utah	32802.00

Figure IV-13: Data computation design for accurately computing total benefits for associate professors

University	Total compensation for associate professors	Total salary for associate professors
Univ_of_Southern_California	265854.00	203196.00
Hollins_College	13824.00	10746.00
Duke_University	108472.00	87016.00
Chatham_College	6036.00	4764.00
Carnegie_Mellon_University	95358.00	78108.00
Univ_of_Texas_at_Austin	262961.00	208964.00
Univ_Minnesota_Twin_Cities	286884.00	219024.00
Trinity_College	5220.00	4344.00
Univ_of_Cinti_Main_Campus	255020.00	198030.00
Univ_of_Illinois_Urbana	298480.00	256360.00
College_of_St_Catherine	23744.00	19557.00
Emory_University	121422.00	96819.00
Northeastern_University	179140.00	136500.00
Bowling_Green_State_Univ	136528.00	101972.00
Vanderbilt_University	104160.00	83520.00
Univ_of_Tennessee_Knoxville	185250.00	147550.00
Texas_A_&_M_University_Main	248868.00	195040.00
University_of_North_Texas	111650.00	89929.00
Michigan_State_University	327522.00	243542.00
Oklahoma_State_Univ_Main	125723.00	101023.00
Antioch_University	6855.00	5220.00
Univ_of_North_Dakota_Main	60636.00	48484.00
Louisiana_St_Univ_and_A&M_C	208949.00	169119.00
Ohio_University_Athens	148938.00	117126.00
University_of_Pennsylvania	199440.00	149280.00
Northern_Arizona_University	99156.00	81004.00
Massachusetts_Inst_of_Tech	142662.00	112749.00
Hofstra_University	90652.00	72050.00
Brown_University	65900.00	51500.00
University_of_Utah	132594.00	99792.00

Figure IV-14: Pure mapping design for accurately computing total benefits for associate professors

In order to perform the task more accurately with mapping techniques, we need to encode both value sets as text as in Figure IV-14. The observational distance for such a design is very high because we need to cognitively compute the *compensation* and *salary* differences for each university. It is much more effective, in this situation, to perform the task with data transform techniques and only present the computed differences to users as in Figure IV-13.

Table IV-1 compares the semantic distance between the pure mapping design (Figure IV-14) and the data computation design (Figure IV-13) based on the dimensions presented in the previous section. Table IV-1 shows that the mapping design has a much greater observational distance. This is because in the mapping design, the user needs to look up each of the text values, thereby resulting in *2n* perceptual lookups where *n* represents the number of universities. Given that the perceptual difficulty of a lookup is *p*, the perceptual load is *2np*. Apart from the perceptual load, there is also a cognitive load (*nc*) for mentally computing the difference between the *total compensation* and *total salary* values for each university (where *c* represents the difficulty of the mental difference operation). Also note that mentally computing differences is significantly more difficult than performing a perceptual lookup so $c >> p$. On the other hand the data computation design only requires a single perceptual lookup for each university, so the observational distance is *np*, which is substantially lower than the observational distance of the mapping design which is $2np + nc$. Thus when a task needs to be performed accurately, we assign a higher cost to the mapping solution and a lower cost to the data computation solution.

	Articulatory		**Expressive**			**Observational**	
	Task Clarification	Navigation	Task	Data	False	Perceptual	Cognitive
Data Design (Figure IV-13)				*2ne*		*np*	
Mapping Design (Figure IV-14)						*2np*	*nc*

Table IV-1: Semantic distance for computing the total benefits for associate professors

Both visualizations do not require any user input (i.e. no articulatory distance). In addition, they are expressive of the difference task and do not show any false information, so their task and correctness expressive distance is nil. However, the data computation design has a greater data expressive distance compared to the mapping design. In the data computation design, the original data has been summarized and it is no longer possible to extract the *total_compensation_associate_professor* and *total_salary_associate_professor* figures from each university. Thus, *2n* facts have been lost.

There are other tasks that require 'fuzzy' accuracy. For example, a person looking for houses in the *Shadyside* area may want to include some houses on the area boundaries even though they may technically fall within other neighborhoods. It is difficult to model such 'fuzzy' accuracy within the computer, and thus the articulatory distance for such tasks are large. Consequently, it is more appropriate to map the data to graphics so users can perceptually determine the appropriate level of 'fuzziness' for the task. This issue also relates to the task specificity issue, which we describe in section IV-3.6.

IV-3.2 Intermediate Tasks

When performing a complex task, we commonly need to break it down into several simpler tasks. For example, suppose we want to determine the total benefits given to full professors as well as associate professors for a set of universities. This operation can be decomposed into two *difference* operations between *total_compensation_full_professor* and *total_salary_full_professor,* as well as *total_compensation_associate_professor* and *total_salary_associate_professor* to get the *total benefits* for each faculty type. An *addition* operation is then applied to the two *total benefits* results as is shown below.

```
(Compute   Addition,
           (Compute   Difference,
                      total_compensation_full_professor,
                      total_salary_full_professor )
           (Compute   Difference,
                      total_compensation_associate_professor,
                      total_salary_associate_professor ) )
```

Task IV-2: Task for determining the total benefits given out to full professors and associate professors.

In Task IV-2, the *difference* operation produces intermediate results that are subsequently used by the *addition* task to produce the final result. As such, the *difference* tasks are not interesting in and of themselves. Tasks whose results are further processed by other tasks are called *intermediate tasks.* Intermediate tasks should be performed with data computation because they simplify the final graphic design by summarizing part of the data and hiding information that does not directly pertain to the main task. This reduces the amount of clutter within the graphic as well as the amount of perceptual interpretation that must be performed, without removing any of the information pertinent to our primary goal. Thus for intermediate tasks, the data solution is given a lower cost than the mapping solution which gets a higher cost in addition to the cost of the extra data attributes that need to be mapped.

For example, consider Figure IV-16 and Figure IV-15, which shows two visualization designs for solving the total benefits task (Task IV-2). In Figure IV-16 (pure mapping design) *total compensation* and *total salary* of each faculty type are mapped to the heights of four bars for each university. To solve Task IV-2 users must compare the lengths of the first two bars to get the *total benefits* for full professors and the lengths of the next two bars to get the *total benefits* for associate professors. This generates a perceptual load of $2p_2$, where p_2 indicates the perceptual cost of each difference comparison. Apart from perceptually estimating the length differences, users must also determine their combined lengths (i.e. the *total benefits* from both faculty types). This results in an additional load of p_3 where p_3 is the cost of estimating the combined length differences and then translating that back into a total benefit value. Thus the total observational load for each university, using the mapping design (Figure IV-15), is $2p_2 + p_3$. These

perceptual operations need to be performed for each university so the total perceptual load is $n(2p_2+ p_3)$, where n is the number of universities.

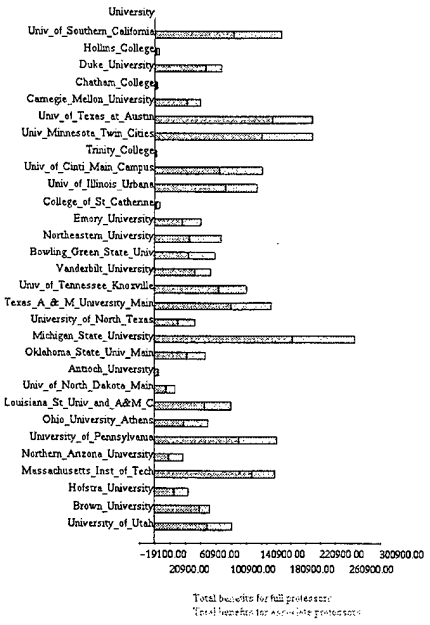


Figure IV-15: Data computation design for computing total benefits for associate professors and full professors. In this case both total benefits have been pre-computed and are shown as stacked bars.

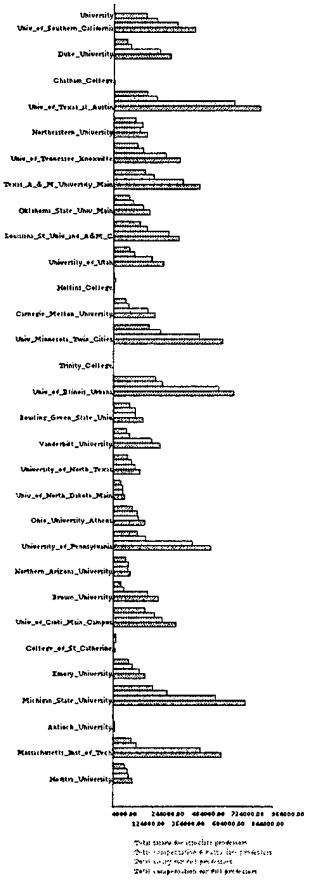


Figure IV-16: Pure mapping design for computing total benefits for associate professors and full professors. In this case, we need to perform the entire task perceptually. Initially we must get the bar differences of the first two bars (red and green) and the last two bars (blue and purple). We must then sum up these differences to get the total benefits.

In Figure IV-15, the *difference* intermediate tasks have been performed with data transform techniques, and the *total benefits* for each faculty type are represented as stacked bars. The total benefits from each university can be determined by simply looking at the height of each stacked bar. In this case, the perceptual load is only np_1, where p_1 measures the cost of a perceptual look up (i.e. looking up the bar length value from the *x-axis*). The cost of estimating total benefit values from two combined length differences in Figure IV-16 (p_3) is clearly more difficult compared to the axis value lookup (p_1), thus $p_3 >> p_1$. The observational load for the mapping design $[n(2p_2+ p_3)]$ is therefore greater than that of the data computation design (np_1).

	Articulatory		Expressive			Observational	
	Task Clarification	Navigation	Task	Data	False	Perceptual	Cognitive
Data Design (Figure IV-15)				*4ne*		np_1	
Mapping Design (Figure IV-16)						$n(2p_2 + p_3)$	

Table IV-2: Semantic distances for total benefits task (Task IV-2)

By performing the intermediate tasks (i.e. difference tasks) with data computation, we reduce the number of values that need to be shown by at least half. Rather than having to show the *total_compensation_full_professor, total_salary_full_professor , total_compensation_associate_professor,* and *total_salary_associate_professor* data attributes (as was done in Figure IV-16), we only show *total_benefits_full_professor* and *total_benefits_associate_professor* in the data computation design in Figure IV-15. Therefore there is less clutter in the display and less output space is required. However, the data expressive distance is also higher because of data filtering. Nevertheless, since the *difference* tasks are intermediate tasks summarizing and hiding their origin data values is appropriate because the tasks are only important for the results they generate in service of the main *addition* task.

IV-3.3 Availability of Perceptual Operations

Certain tasks can be easily offloaded onto the perceptual system without adding much, if any, observational distance. Some examples are *addition* and *subtraction*, which can be mapped to *stacked bars* and *overlapping* or *interval bars* respectively. In each of these cases, the task results are perceptually summarized onto one graphical feature. The results of the addition task are summarized by the stacked bar *heights* and the results of the subtraction task are summarized by the interval bar *lengths*. Certain abstract mathematical tasks (e.g. logarithmic or exponential computation) do not have any perceptual parallels and cannot be offloaded onto the perceptual system. Such tasks also tend to have high cognitive loads, which results in large observational distances. For such tasks, data computation techniques can be used to offload the expensive cognitive computation onto the computer system.

Other tasks such as summarization tasks (e.g. *sums*, *mean*, and *median*), or getting the *minimum* and *maximum* values within a set, can be performed perceptually but require more perceptual effort from users compared to the *addition* and *subtraction* tasks. For example to find the maximum data value from a bar chart we would need to compare the *heights* of a set of bars and pick the tallest one. Unlike the *addition* and *subtraction* cases, the task result is not captured in a single perceptual value but rather has to be derived by considering a set of perceptual values.

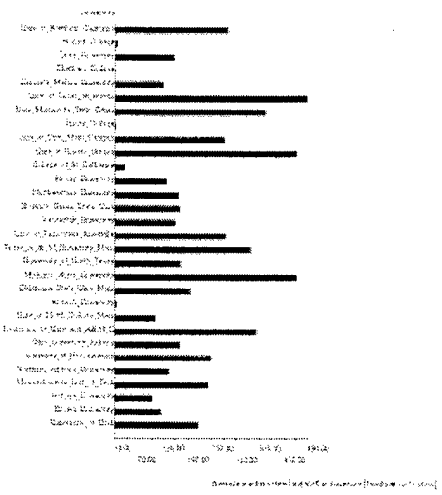


Figure IV-17: Data computation design for computing average number of teaching staff per university. In this case the average number of teaching staff has been pre-computed and the results are shown on the *x-axis*.

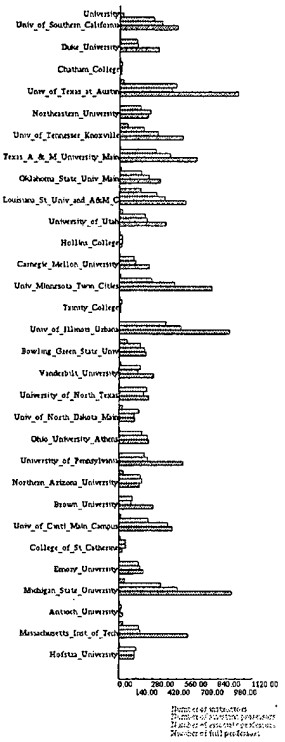


Figure IV-18: Pure mapping design for computing average number of teaching staff per university. In this case the average number of teaching staff must be perceptually estimated by finding an average line across each cluster of bars.

For example, suppose we want to determine the average number of teaching staff (including *full professors, associate professors, assistant professors,* and *instructors*) within a set of universities.

	Articulatory		Expressive			Observational	
	Task Clarification	Navigation	Task	Data	False	Perceptual	Cognitive
Data Design (Figure IV-17)				$4ne$		np_1	
Mapping Design (Figure IV-18)						$x(4np_2 + p_3) + np_1$	

Table IV-3: Semantic distance for finding average number of teaching staff for a set of universities

The pure mapping design (Figure IV-18) maps the number of each faculty type to a differently colored bar length. To estimate the average number of teaching staff, we pick a line that separates the four bars in such a way that the sum of bar lengths above the line is equal to the sum of lengths below it. This requires at least four bar difference estimations (to get the lengths above and below the average line) ($4p_2$)

and several comparisons between the lengths on top and below to determine their equality (p_3). When the lengths above and below are **not** equal we must re-estimate a new average line and repeat the process above (x). Once we get an acceptable average line we can look up the average value from the *y-axis* (p_1). The total perceptual load is therefore $x(4np_2 + p_3) + np_1$.

On the other hand, the data computation design (Figure IV-17) pre-processes the average number of teaching staff and only maps the results to bar lengths. Thus, users only need to perform n bar length lookups resulting in a perceptual load of np_1. The observational distance for Figure IV-18 (the mapping design) is much larger because there are more visual artifacts that must be attended to (4 bars instead of just 1) and because of the perceptual load needed for estimating and re-estimating the mean number of teaching staff (x). Thus for tasks that have good perceptual parallels (e.g. *addition*), the data and mapping designs are rated equally by our automatic design system. On the other hand, for tasks that **do not** have any perceptual parallels (e.g. *exp, log*) or for tasks that have high cost perceptual representations (e.g. summarization tasks), preference is given to the data computation solution.

IV-3.4 All to All Operations

Thus far, we have been considering tasks that compute or compare pairs of values, e.g. computing the difference between *total compensation* and *total salary* for each university faculty type. These pair-wise (*value-pair*) comparisons occur very commonly in data analysis, but do not represent the *only* task class. Another important class of tasks is all-to-all tasks. All-to-all tasks require each value in a set to be processed with all values in the second set, e.g. processing the *total compensation* values for each university with the *total salary* values of all other universities.

For a more realistic all-to-all task, consider an extended airline-scheduling task analogous to the one described in the airline-scheduling task in section IV-1 (Task IV-1), except here we take both *total-cost* and *total-downtime* into consideration.

> *"Given an origin and a destination city, the user "attempts to locate the two flights arriving in and departing from a layover city that offer the minimum amount of* ***cost*** *and* ***`down time'*** *between the flight times and the beginning and ending time of a scheduled meeting (in the layover city)".*

As before we assume that the origin city is *Los* Angeles, the layover city is *Chicago*, and the destination city is *Boston*. In addition, the meeting in *Chicago* is from 2 p.m. to 4 p.m.

This is an *all-to-all* task because we must compare all flights before the meeting with all flights after the meeting. To solve this task with data computation we pre-process the total downtime and total cost for all flight pairs as in Figure IV-19 (total downtime is encoded with *x-length* and total cost is encoded with *saturation*). If there are n_1 flights before the meeting and n_2 flights after the meeting, we must calculate and show values for $n_1 * n_2$ flights (i.e. $O(n^2)$ flights, where n is the total number of flights in the data set).

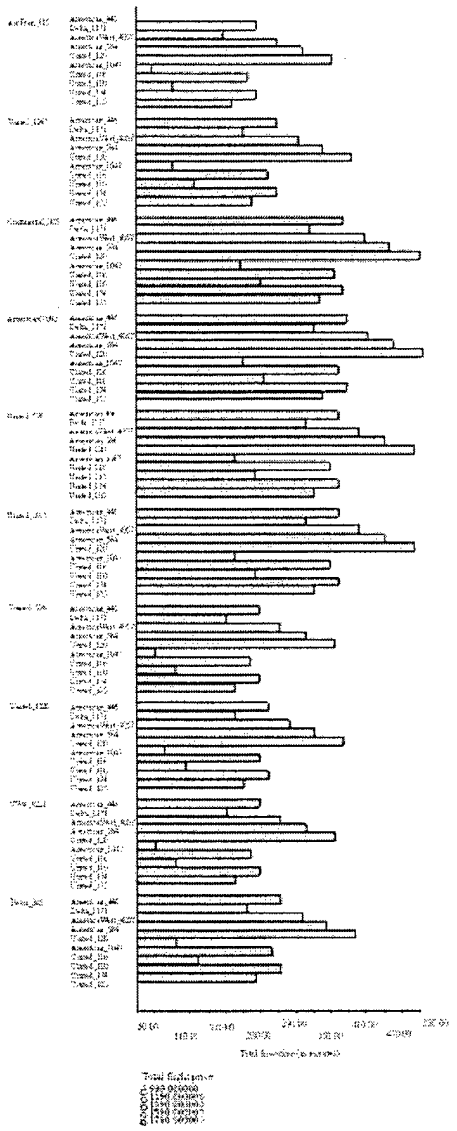


Figure IV-19: Data computation design for computing total downtime and total cost for all pairs of flights that fulfill our airline-scheduling criteria. Total downtime is pre-computed and encoded on the *x-axis* while total cost is pre-computed and encoded as *saturation*.

Although Figure IV-19 only contains information on 20 different flights, (10 * 10) =,100 values must be shown because of the *all-to-all* comparison and it is difficult to display all the information together clearly on the CRT screen. Therefore, we may need to navigate around the visualization space during the analysis session. Given that we have enough space on the CRT screen to show x elements, we need to scroll the visualization $O(n^2)/x$ times in order to get to all the information (i.e. there are $O(n^2)/x$ information slices). Each scroll requires moving the mouse over to the scroll bar (m), a mouse click on the scroll bar control (k), moving the scroll control (m) and a mouse release (k). The articulatory load is therefore $O(n^2)/x * 2(m+k)$ where m is the cost of a mouse move and k is the cost of a mouse click or release. To find the pair

of flights with the best balance between total downtime and total cost, we choose the shortest bar with the lowest saturation in each information slice and compare these bars across slices to get the best one. This results in a perceptual load of $(O(n^2)/x) * p_1$ where p_1 indicates the perceptual cost of each bar search and comparison and n represents the number of flights.

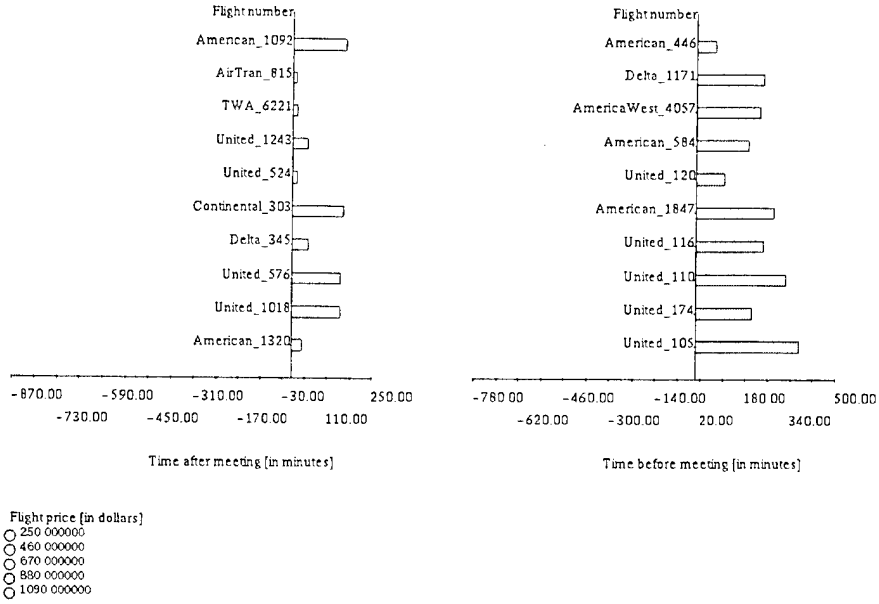


Figure IV-20: Mapping design for computing total downtime and total cost for all pairs of flights that fulfill our airline-scheduling criteria. *Time_after_meeting* is mapped on the *x-axis* of the left chart, *time_before_meeting* is mapped on the *x-axis* of the right chart, and *flight_price* is mapped to *saturation* in both charts.

Figure IV-20 shows the mapping techniques solution for solving the same task. In this design, the total cost and total downtime computations are **not** performed with data techniques. Instead, separate cost and downtime information are shown on both legs of the flight. The downtime and cost for the first leg of the flight is shown as *x-length* and *saturation* on the left chart in Figure IV-20 and similar information on the second leg is shown on the right chart. In order to find flights with low cost and low total downtime, we look for shortest, least saturated bar in each chart. Assuming that we can display x elements in the given amount of space, the navigation load would only be $(O(n)/x) * 2(m+k)$ because we only need to show at most $n_1 + n_2$ flights compared to the $O(n^2)$ flights in the data computation design shown in Figure IV-19. To find the best flights using Figure IV-20 we must look for two of the shortest and least saturated bars in each information slice, thereby resulting in a load of $2p_1$ for each slice. Total perceptual load therefore, comes to $O(n)/x * 2p_1$ where n is the number of flights, x is the number of elements that can be displayed on the CRT screen, and p_1 measures the difficulty of locating the shortest, least saturated bar in each information slice and comparing that bars across information slices

	Articulatory		Expressive			Observational	
	Task Clarification	Navigation	Task	Data	False	Perceptual	Cognitive
Data Design (Figure IV-19)		$O(n^2)/x$ * $2(m+k)$				$O(n^2)/x\ p_1$	
Mapping Design (Figure IV-20)		$O(n)/x$ * $2(m+k)$				$O(n)/x\ p_1 + 3p_2$	

Table IV-4: Semantic distance for an airline-scheduling task which balances total downtime and total cost

Based on Table IV-4, we see that the mapping design is far superior because when we try to solve an all-to-all task with data computation, we are forced to show many more data values, and this results in greater navigation and perceptual loads. Although in the mapping design there is the additional cost of having to process two charts, this cost is far outweighed by the processing needed for the large number of objects in the data computation design. I.e. for larger n the $O(n^2)/x$ factor in the data computation design far outstrips the $O(n)/x * 2$ factor in the mapping design. Although we have assumed that cognitive load is negligible, it can be fairly significant here, because we must compare data sets across several separate screens and as a result, we may need to maintain some context in short term memory across different information slices. Since there are more data slices that we need to traverse in the data computation solution, the related cognitive costs will probably be greater as well. Thus, for all-to-all tasks, our designer assigns a lower cost to the mapping solution particularly if the data set is large and if there are effective graphical representations for showing the task.

IV-3.5 Task Variation on Attribute

A big disadvantage of using data computation techniques to solve tasks is that they are limiting, i.e., they serve very specific purposes and cannot be adapted for a wide range of different goals. When performing a task through data computation we only show the results of the computation and hide the initial and intermediate values from users. Consequently, the resulting visualization design can only be used to solve its original, intended goal. For example, if we used data computation to perform an *addition* task we cannot also perform a *difference* task based on the computed results because the original values have already been summarized. Data simplification comes at the cost of inflexibility.

People, however, are much more versatile, and by mapping the source data onto graphics, we give end users greater flexibility in being able to solve a wider range of tasks with the same graphic. Thus when we need to solve a set of different tasks that operate on the same data attributes, we often end up having a higher observational distance if we use data computation operations. This is because a new set of values must be computed and visualized for each task variation. When the data complexity added by the data computation is greater than the cognitive load it subtracts, we should address the task by using mapping techniques.

Suppose we are studying election data for three different political groups over multiple states. Our task is to view the total number of votes in each state to determine its importance as well as to rank the political groups based on their individual number of votes. To fulfill this task with data computation (Figure IV-21) we process and represent the total number of votes by using the length of horizontal bars and align each bar with a pre-computed ordered list of the three political parties. This ordered list is represented by a series of dots, ordered from left to right, with each dot representing a different party. Color is used to encode the party type. In order to solve the task we look up the total votes from the *x-axis* (np_1) and the ranking information from the series of aligned dots (np_2).

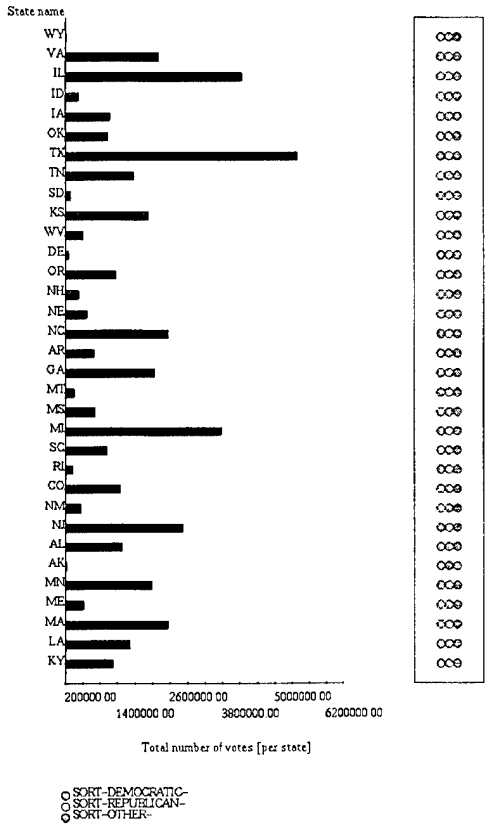


Figure IV-21: Data computation design for computing total number of votes in each state and ranking the three political parties based on the number of votes received. *Total_number_of_votes* has been pre-computed and is shown on the *x-axis* of the left chart. *Party_ranking* has also been pre-computed and is shown in the right table.

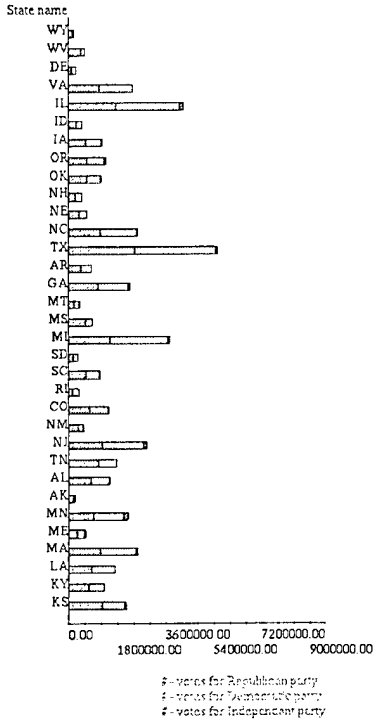


Figure IV-22: Mapping design for computing total number of votes in each state and ranking the three political parties based on the number of votes received. The *#_votes_for_Republican_party* is mapped to the *x-length* of the *red* bar, the *#_votes_for_Democratic_party* is mapped to the *x-length* of the *green* bar, and the *#_votes_for_Independant_party* is mapped to the *x-length* of the *purple* bar. Total votes can be derived by looking at the combined length of the stacked bar and party ranking can be derived by comparing the three differently colored bar lengths for each state.

Alternatively we could represent the task with mapping techniques by combining the number of votes received by each political party as a stacked bar (Figure IV-22). We can deduce the total number of votes as well as the group ranking from the same graphical representation (i.e. both tasks can be performed using the same graphical objects). We can look up the total number of votes from the combined height of the stacked bar (np_1) and determine the party ranking by comparing the lengths of the different divisions within each stack (np_3).

In this example, the mapping design (Figure IV-22) is preferable. Both designs have comparable observational distances; however, the mapping visualization is much more expressive. As is shown in Table IV-5, the data computation design has a *data expressive* distance of *3ne* (where *e* represents the expressive distance for each inaccessible number of votes figure) because we can no longer derive the original number of votes for each political group from the summarized results.

	Articulatory		**Expressive**			**Observational**	
	Task Clarification	Navigation	Task	Data	False	Perceptual	Cognitive
Data Design (Figure IV-21)				*3ne*		$np_1 + np_2$	
Mapping Design (Figure IV-22)						$np_1 + np_3$	

Table IV-5: Semantic distance for finding the total and individual sales

The data computation design has more clutter and shows less information compared to the mapping design without cutting down the observational distance of the task. This is because we were able to achieve two different tasks using the same graphical objects in the latter case while in the former case we had to encode the results of each task using two different sets of objects. Generally, when there are good perceptual parallels and significant task variation over the same data attributes, our automatic design system favors a mapping solution over a data computation design. When no effective perceptual parallels are available however, then our system weighs the cost of having more clutter (i.e. more graphical objects) and lower data expressiveness in the data computation design with the added cost to cognition and perception from having to perform the task perceptually with mapping techniques.

IV-3.6 Task Specificity

Tasks can be stated at many different levels of specificity. The higher the level of specificity, the cheaper it is to accomplish the task with data computation. When tasks cannot be fully specified at the outset, users must supply the missing task arguments to the data computation functions during the analysis process. Consequently, users are required to learn and use a set of input devices and interface artifacts, which increases the articulatory distance of the design. For example, in the airline-scheduling task

presented earlier, we must know the *origin, destination,* and *layover* cities as well as the meeting time, in order to use data computation to solve the task. If we are unsure of these task parameters, we must supply then during analysis with input devices, causing a higher articulatory distance.

Sometimes, tasks cannot be described with high specificity because it is difficult to capture the task requirements or constraints. For example, suppose we want to find a "good" university to attend. We would like the university to have relatively low tuition cost, but a good record of accomplishment for graduating its students, and a good student/faculty ratio (i.e. low ratio). We might be willing to pay more tuition however if the university has an exceptionally high graduation rate or low student/faulty ratio. In general, we want to pick a university based on a balance of all three factors. Note that for this task it is difficult to specify the input parameters fully because there is no "correct" set of parameter values.

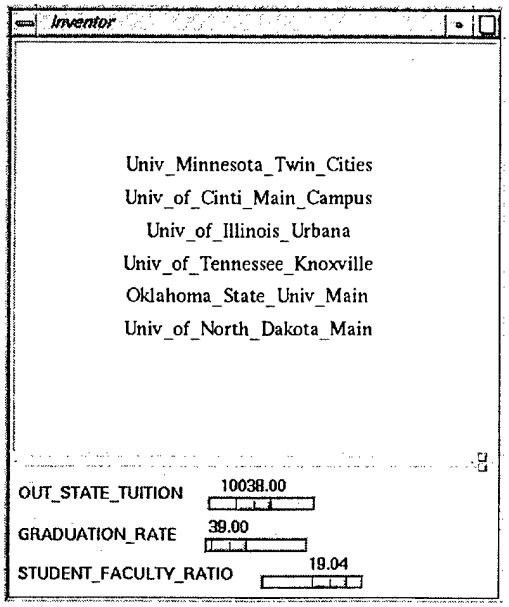


Figure IV-23: Data computation design for finding the best university based on *out-of-state-tuition, graduation-rate,* and *student-faculty-ratio.* Thresholds for each condition can be entered through the three sliders and those universities that fulfill the threshold conditions are pre-computed and shown.

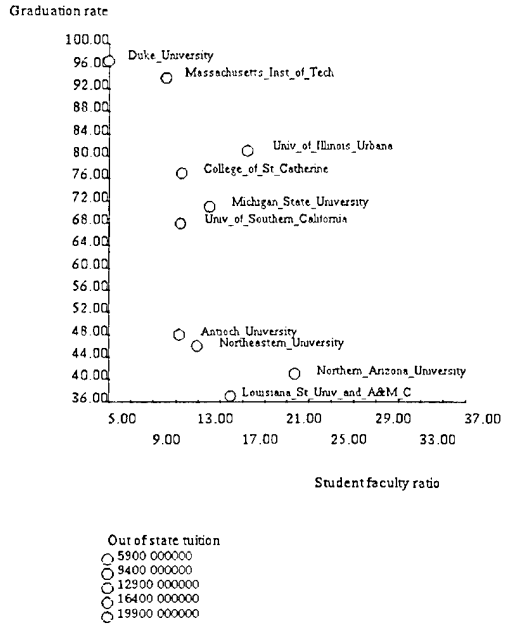


Figure IV-24: Mapping design for finding the best university based on *out-of-state-tuition, graduation-rate,* and *student-faculty-ratio. Student-faculty-ratio* is mapped to the *x-axis, graduation_rate* is mapped to the *y-axis,* and *out_of_state_tuition* is mapped to *saturation.* The best universities are those in the upper-left corner of the display, with low *saturation.*

In order to solve this task with data computation, we need to try out different parameter value combinations by entering them into the system using input devices. Figure IV-23 shows such an interface. It has three sliders for indicating the acceptable *tuition*, *graduation rate*, and *student/faculty* thresholds. The

articulatory distance for this design depends on the number of parameter entries we must make. At the very least, we must perform three input device manipulations to feed in the initial threshold values. Each entry requires the user to place the mouse over the slider, click on the slider controller, drag the controller to the correct position, and release the mouse. This produces a load of $2(m + k)$ for each entry where m indicates the load incurred for a mouse move and k indicates the load for a mouse click or release. The minimum articulatory load for Figure IV-23 is then $6(m + k)$. Once we have entered these threshold values, all the universities that fulfill our constraints are shown as labels. Suppose that for each category half the universities pass our query, this would result in $n/8$ universities, thus the perceptual load for reading the resulting university names is $(n/8)\ p_1$.

Figure IV-24 shows how we can solve the same task with mapping techniques. In Figure IV-24, each university is represented as a *labeled mark*. *Student/faculty ratio* is mapped to the *x-axis*, *graduation rate* is mapped to the *y-axis*, *tuition cost* is mapped to the *saturation* of the *marks*, and the *university name* is mapped to *labels* next to each *mark*. Universities that fulfill our task criteria can be found by looking to the top-left corner of the chart (high *graduation rate,* low *student/faculty ratio*). We may slightly relax our constraints and consider the adjacent areas which indicate universities that have either lower *graduation rates* but good *student/faculty ratios* or high *graduation rates* but weaker *student/faculty ratios*. Within each of these areas, we are only interested in the less saturated *marks*, which indicate universities with lower tuition cost. Both the location and saturation lookups are perceptually pre-attentive and thus only two perceptual operations are required to find the appropriate universities (perceptual load of $2p_2$). Once we have identified the universities of interest, we lookup their names from the labels next to each mark. Assuming a uniform distribution, a quarter of the objects will be in the area that we are considering (i.e. $n/4$). In addition, we are only interested in the universities with lower tuition cost, i.e. the less saturated marks. Assuming that half of the universities in our area of interest are less saturated, we must attend to $n/8$ university labels. Thus, the total perceptual load for this design is $(n/8)\ p_1 + 2p_2$.

	Articulatory		Expressive			Observational	
	Task Clarification	Navigation	Task	Data	False	Perceptual	Cognitive
Data Design (Figure IV-23)		$6(m+k)$		$3ne$		$n/8p_1$	
Mapping Design (Figure IV-24)						$n/8\ p_1 + 2p_2$	

Table IV-6: Semantic distance for finding a house based on price, size, and distance to workplace

Based on Table IV-6, the mapping design is superior to the data computation design because the mapping design has no articulatory load and the additional two perceptual lookups required do not add much to the observational distance. For larger data sets, however, the mapping design could become cluttered and objects may be occluded (Figure IV-24). In this case, we would need to use input devices for

navigation purposes, and semantic distance may end up being higher compared to the data computation design.

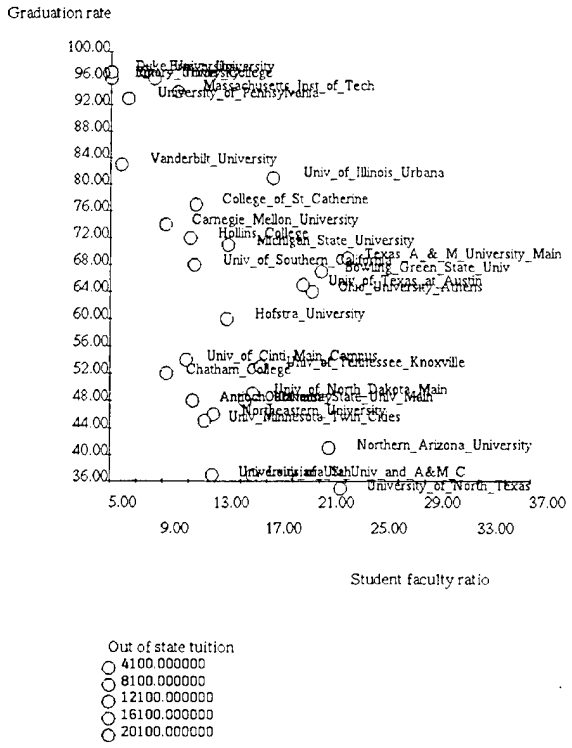


Figure IV-25: Identical design as Figure IV-24 but applied to a larger data set. As a result there is significantly more occlusion making it difficult for us to accurately view the *saturation* values on the *marks* as well as read the university names.

IV-3.7 Summary

In this section, we presented a set of high-level design guidelines that can help us determine whether to solve a task by mapping its data to graphics or by pre-processing it using data transforms. Details on how these more abstract design rules can be translated into concrete heuristics for an automatic visualization design system (AVID) are described in chapter V. Based on the guidelines presented here, we show how these data and mapping designs may be ordered according to their effectiveness at solving current user goals. In appendix C-4 we systematically explore the range of tasks available in our framework and describe how changes to the task will affect the design choices made by our automatic design system based on the design dimensions and guidelines presented in this chapter. In appendix C-5 we explore the possible space of data and mapping designs for a car purchasing task. For a complete description of our task language refer to appendix C-3.

IV-4 Conclusion

In this chapter, we described a set of design guidelines (section IV-3) that can be applied in an automatic design system for making decisions between using data transforms and mapping transforms. These design guidelines reduce the semantic distance for solving a task by either reducing articulatory distance, expressive distance, or observational distance. Designs that have many data computations usually have a smaller observational distance because some of the perceptual and cognitive load of interpreting the graphic has been offloaded onto the computer system and only the summarized task results are shown. On the other hand, data computation designs require complete task specifications (i.e. no task unknowns), which usually results in a greater articulatory distance. In addition, data computation designs also reduce the expressive capabilities of a visual design by filtering out all data that is not absolutely pertinent to the task. As a result, the range of perceptual tasks enabled by the design is reduced. Thus, when there is significant task variability over the same data attributes a mapping design is preferred.

High level user analysis goals generally consist of a mix of well specified subtasks (where it is clear what the goals of the task are and what the task parameters are) and non specific subtasks. Thus an effective design will most likely consist of a combination of data and mapping transforms. The blend of data and mapping transforms that is most appropriate is based on the interaction among the input task, data set, available graphical representations, as well as input and output hardware. Because the design decisions are based on a wide range of factors, it can sometimes be difficult to decide which sections of a task are more suitable for data computation and which are more suitable for mapping transforms. In this chapter we present a set of design guidelines that can help guide designers in making these decisions. These guidelines can also be translated into design heuristics and included into an automatic design system. We showed in section IV-1 that including data computation operations into the automatic design process significantly expands the visualization design space and the effectiveness of the system in being able to deal with data analysis problems. In appendix D, we analyze three more example tasks and systematically show the new set of designs that our work enables over previous research in this area. We also show that the rankings made by our system based on the design guidelines presented in this chapter conform to GOMS estimated performance time. Specifics on the architecture of our automatic design system are presented in chapter V.

Chapter V: Implementation

Automatic Visualization Interface Designer

In appendix E we evaluated a set of visualizations generated by our automatic design system, AVID (Automatic Visualization Interface Designer). The evaluation results (appendix E) show that expanding our understanding and vocabulary of visualization primitives to include data computation/transformation operators, perceptual or mapping transform operators and input device components, can enhance our ability to generate visual designs that are interesting and appropriate for our information tasks. This chapter describes how our automatic design system, AVID, is implemented based on the visualization functions framework described in chapters II and III and the visualization design heuristics and metrics described in chapter IV. The implementation of AVID shows that the theoretical concepts we developed previously are complete and specific enough to be applied to a real system. This chapter also highlights the system engineering issues that must be considered to capture the new function classes and heuristics we introduce in our work. Note that all the visualizations shown in this document are generated by AVID unless otherwise noted.

AVID, consists of three components corresponding to the three stages of the automatic design process (shown in Figure V-1):

1. *The task specification component*: Initially, a higher level agent (user or a domain specific system) that has a deeper understanding of the problem domain generates a set of tasks for AVID. Tasks are expressed using a simple language based on the EDA (Exploratory Data Analysis) task model first developed by Tukey [*Tukey, 1977*] and later refined by Casner [*Casner, 1991*] for automatic design. This language is relatively low-level and its purpose is to capture important components of a task that may affect the visual design process. We do not expect typical end users to specify tasks in this language; rather, specifications will most likely be generated by domain specific systems that use graphics to present and summarize their results to users, such as automatic planning systems, automatic information analysis systems, agent based information gatherers, etc. We described general concepts of our task language in appendix C-3. In this chapter we discuss the implementation details of the language and how it is interpreted by AVID. Specifically AVID deals with processing embedded tasks as well as *accuracy* and *iterative* special task conditions that are not dealt with in previous automatic systems but are crucial in our work because of their impact on data transform functions and input devices.

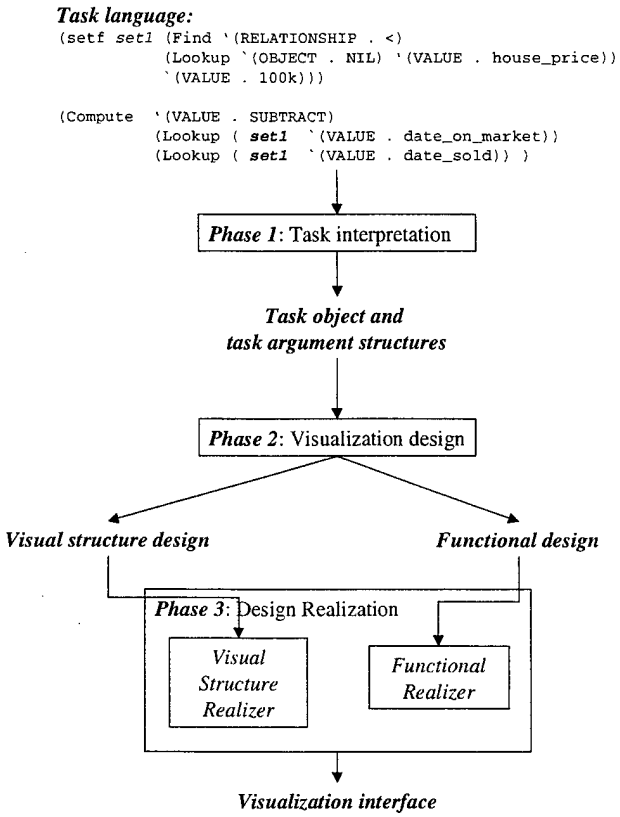


Figure V-1: Three components within AVID that correspond to the three stages in the automatic design process: 1) Task interpretation, 2) Visualization design, and 3) Design Realization

2. *The design component*: In the design component, AVID parses task objects and argument structures generated from the task interpretation component and converts them to design constraints and preferences. Based on these constraints, AVID explores the design space for the input tasks and automatically generates a set of visualizations ordered from best to worst. These output designs are expressed in a language that captures the visual structure of a visualization interface as well as any underlying transform functions and active interactive components. Visual structure descriptions have been developed in previous work [*Mackinlay, 1986a, 1986b; Chuah, 1995*]. This thesis develops a language for capturing the functions and active components within a visualization (described in chapters II and III).

One of the main contributions of our designer is in expanding the visualization design space to include data transforms, mapping transforms, and interactive components (i.e. input devices). Our GOMS evaluation tests in appendix E showed that this expansion allows us to more effectively address the class of analysis tasks considered in this work compared to previous automatic designs that solely rely on mapping operators. The expanded design space enhances human computer communication because a greater visual vocabulary allows more efficient communicative

constructs to be generated. In addition, the effectiveness of AVID as a design assistant is also increased because it is able to provide more design alternatives and choices to users. To enable this expansion in the visualization design space, AVID incorporates new procedures in its search algorithm over what has been done in previous systems. Specifically, previous systems only considered how data attributes can be effectively mapped to graphical properties in a *data attribute mapping* procedure (section V-2.1.2). In AVID, we have an additional *task processing* procedure (section V-2.1.1) that considers whether to apply data transforms or mapping transforms to solving tasks, what hybrid data and mapping transforms are valid design alternatives, as well as how to address embedded tasks, object filtering, and unknown task arguments. Our design system also culls out bad designs (i.e. task inexpressive designs or designs that do not support the input task(s)) as well as duplicate designs (section V-2.1.3). This saves users from having to devote attention to these less appropriate visual representations while still having good coverage of the design space. This issue was also not considered in previous systems.

3. *The realization component*: The "realizer" component interprets design specifications generated by the design component and renders an active visualization interface. This component makes layout decisions and assigns default values to visual components that are left unspecified or unconstrained in the design specifications. Currently, AVID's realizer is capable of interpreting most of the selection, transformation, and translation functions described in this thesis (e.g. computations, set-operations, threshold operations, etc). By combining these primitives it can render a wide range of interactive behaviors such as aggregation, painting, dynamic queries, simple semantic zoom, SDM graphical manipulation operations [*Chuah, 1995*], navigation operations, etc. Previous systems could not render designs with data transform functions or designs that contain interactive components.

In the following sections of this chapter we describe how our automatic design system, AVID is implemented and how the concepts laid forth in the previous chapters are captured within its three primary components.

V-1 Task Interpreter Component

The task interpreter component accepts task descriptions as input, analyzes the tasks and their arguments for validity, and then produces a set of *task-class* (Figure V-4) and *task-argument* (Figure V-3) structures. These structures are passed to the design component that uses the information to guide its design strategy. The task interpreter accepts specifications that are in LISP form. An example task specification is shown in Figure V-2.

In this task we calculate the duration that houses under 100k in price, stay on the market. Each task within the specification has three parts: the *task class*, the *task input arguments*, and any special *task conditions*. For example, the top task in Figure V-2 can be decomposed into a *find* task class, a list of three task arguments, with no special task conditions. The bottom task in Figure V-2 can be decomposed into a *compute* task class, a list of three task arguments, and two task conditions. The embedded *lookup* tasks within the bottom *compute* can be decomposed in the same way.

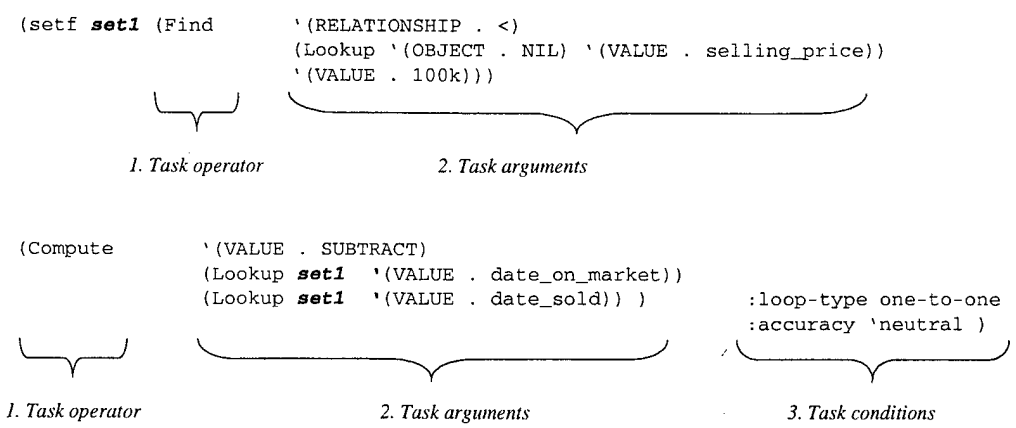


Figure V-2: Example task specification

The three task parts encapsulate the following information:

1. *Task class*: The task class captures the main goal of the current analysis operation. Different task classes require different numbers and types of input arguments as is summarized in Table V-1. Currently AVID can interpret five different task classes: *lookup, find, AND, compare,* and *compute*. These task classes can commonly be used to describe problems that arise in data analysis [*Casner, 1991; Senay, 1994*].

2. *Task arguments*: Task arguments may specify single or sets of values. For example the arguments `'(VALUE . SUBTRACT)`, and `'(VALUE . 100k)` in Figure V-2 contain single values (*subtract* and *100k*). On the other hand the argument `'(VALUE .'(1 2 3 4 5))` contain a set of five values. Task arguments may come in one of two forms:

   a) *Associative value pairs*: An associative pair has two elements (e.g. `'(VALUE . SUBTRACT)`), the first describes the task argument type (e.g. *value*) and the second contains the actual argument value(s) (e.g. *subtract*). Currently we have three types of arguments: *value*, *object*, and *relationship*. *Value* arguments may be numbers or strings; *object* arguments refer to conceptual structures within the visualization such as a *data concept*, a *graphical object*, or a *chart region*; and *relationship* arguments describe relations that may exist among objects or values. Currently, we only consider simple value relationships such as >, <, and =.

   b) *Output results from other tasks*: Tasks may also accept output argument structures that are generated by embedded tasks. The compute task in Figure V-2, for example, accepts results from two embedded *lookup* tasks. Each of these *lookup* tasks produces a set of values corresponding to the *date_on_market* and *date_sold* attributes of the *house* data concepts associated with the visualization.

3. *Task conditions*: Apart from the regular task input arguments, we may also specify special task conditions. Currently, AVID can process two types of conditions, namely the *task loop type* and the *task accuracy level*. Details on these two task conditions can be found in appendix C-3.2.

The *task loop type* describes how the task input value sets should be iterated over. There are three iteration types in our task framework:

a) *One-to-one* is the default iteration type. It specifies that each value in the first set is processed with the corresponding values in all subsequent sets (e.g. the *1st* value in each set are processed together, and so are the *2nd*, *3rd*, *4th*, and *n-th* values);

b) *All-to-all* tasks require each value in an input set to be processed with each and every value in the subsequent sets.

c) *Previous-pair* tasks order the input value sets based on an ordering attribute, then applies each consecutive pair of values within the ordered set to the task function.

The *task accuracy level* describes the level of accuracy that is desirable for the task. Currently, there are three accuracy levels, *accurate, neutral* and *fuzzy*.

AVID's task interpreter possesses a LISP function corresponding to each task class (i.e. *lookup, find, AND, compare,* and *compute*). When activated, each LISP function analyzes the task input arguments to ensure their validity. This includes number-of-argument checks, argument-type checks, and argument-correspondence checks. The number and type of arguments required for each task class is summarized in Table V-1.

	Input arguments	Output argument types
Lookup	• *1 object* argument containing the set of objects to perform the lookup on. *E.g.* `'(object .(house-1 house-2 house-3))` Note that an empty object set defaults to all objects in the database. *E.g.* `'(object . nil)` • *1 value* argument containing the lookup attribute name. *E.g.* `'(value . date-on-market)`	• *1 value* argument
Compute	• *1 value* argument containing the compute operator to apply (e.g. *add*, *subtract*, etc). • *n value* arguments containing the data value sets to compute.	• *1 value* argument
Find	• *1 relation* argument containing the find relationship to apply (>, <, =, >=, <=). • *2 value* arguments containing the data value sets to search on.	• *1 object* argument
AND	• *n object* arguments containing the object sets involved in the *AND* relationship.	• *1 object* argument
Compare	• *2 value* arguments containing the data value sets to compare.	• *1 relation* argument

Table V-1: Task inputs and outputs

Argument-correspondence checks ensure that the input data types are consistent with the task. For example, only arguments of the same type can be added or subtracted from each other. It is not possible to perform *additions* and *subtractions* on a set of *price* data values and a set of *weight* data values. The same correspondence constraint applies to *find* and *compare* task classes.

When a task is specified correctly (i.e. all its arguments are valid), the task LISP function generates:

1. *Output argument structures*: The output arguments generated have the same form as any input *task argument* structure (an example is shown in Figure V-3). It captures properties of the task results that are derived from the task class and the task input arguments. Association value pair arguments (e.g. `'(VALUE . SUBTRACT))` are converted into task argument structures based on the pair values and the argument's parent task. For example, a value pair argument in a *lookup* task (e.g. `'(VALUE . selling_price))` implies that the second value in the pair (*selling_price*) is an attribute name. On the other hand, a value pair in other task classes may imply a data value (e.g. `'(VALUE . 100k))` or a relationship value (e.g. `'(VALUE . <))` depending on the expected input arguments of the task class (as is shown in Table V-1). The output argument structure may be passed on to other tasks as inputs, which is what occurs when we embed one task within another. Input arguments from embedded tasks are already in the desired argument structure form (as was described above) and thus need not be further processed. For example in Figure V-2, the *find* task generates a task argument structure containing a set of data concepts and passes that on to the *lookup* tasks which extract the *date-on-market* and *date-sold* values from those concepts. These two sets of values are subsequently passed on to the *compute* task. Sometimes these output structures may contain newly generated derived or summarization attributes (within its *content* slot) that are used to store the results of a data computation function. For example, the *find* and *AND* tasks generate a boolean attribute (attribute containing T or F values), the *compute* task generates a value summarization or derived attribute, and the *compare* task generates a relationship attribute (attribute containing >, <, or = values).

```
(defclass task-argument (primitive-object)
   ( class          ;; Argument type: OBJECT | VALUE | RELATION
     parent         ;; all tasks that contain this argument
     within         ;; task which produces this task argument
     content        ;; Data attribute or value(s) associated with argument
     viz-function   ;; Internal function used to process results for this argument
    ))
```

Figure V-3: Task argument structure

2. A *task class* structure: An example *task class* structure is shown in Figure V-4. Each task within the input specification is translated into a *task class* structure. All input argument structures associated with the task are collected and placed within the *arg-list* field slot.

```
(defclass task-class (primitive-object)
   ( class          ;; task class: either [ LOOKUP, COMPUTE, COMPARE, FIND, AND ]
     arg-list       ;; input task arguments
     is-embedded    ;; whether task is embedded within another [ t | nil ]
     accuracy       ;; task result accuracy [t = accurate, nil = approximate]
     loop-type      ;; Loop method on input objects[one-to-one|all-to-all|previous]
     num-times      ;; task frequency: number of times a task is to be repeated
     output-arg     ;; output task argument structure produced by task
    ))
```

Figure V-4: Task class structure

V-2 Automatic Design Component

Our automatic designer is implemented using Common LISP (Allegro version 4.0), and a constraint satisfaction system called SCREAMER [*Siskind*]. The designer accepts a list of task class structures (shown in Figure V-4) and task argument structures (shown in Figure V-3) as input and produces a set of design specifications as output, ordered according to task effectiveness. Each design in the output set fulfills all the input task requirements. A design specification consists of two components: a) a structural description of the graphical components within a visualization and b) a description of the functional components within a visualization (this functional specification corresponds to the framework language described in chapters II and III of this thesis).

In the following sections we outline our strategy for exploring the space of visual elements and visualization techniques as well as describe how the heuristics provided in chapter IV can be encoded as design constraints and design costs. The constraint and cost structure directs the search algorithm and allows the AVID design component to generate an ordered list of designs that reflect cognitive, perceptual, and articulatory complexity with respect to the input task(s).

V-2.1 Search Strategy

AVID's search procedure has two primary phases: the *task processing* phase, and the *data attribute mapping* phase. These two phases are indicated on the search strategy flowchart in Figure V-5. The section to the left describes the *task processing* phase, the section to the right describes the *data attribute mapping* phase. The *data attribute mapping* phase is what was performed in previous automatic systems. **To enable the design of visualizations that contain data transforms and input devices, we added the *task processing* phase**. The *task processing* phase begins with the first outermost task and then proceeds to all embedded tasks within it. Consider the house task described previously (Figure V-2).

In this task we are interested in seeing whether houses costing less than 100k stay on the market for relatively short periods of time. The task specification is shown again below:

```
(setf set1 (Find        `(RELATIONSHIP . <)
                        (Lookup `(OBJECT . NIL) `(VALUE . selling_price))
                        `(VALUE . 100k)))

(Compute        '(VALUE . SUBTRACT)
                (Lookup set1  '(VALUE . date_on_market))
                (Lookup set1  '(VALUE . date_sold)) )        :loop-type one-to-one
                                                             :accuracy nil )
```

Before the task processing phase begins, AVID's design component orders its input set of tasks according to their embedding structure, from outermost to innermost. Based on this ordering method, the house task described above would be organized as follows (the numbers in the angle brackets "[]", indicate embedded tasks):

```
1. (Compute '(VALUE . SUBTRACT) [2] [3])
2. (Lookup [4] '(VALUE . date_on_market))
3. (Lookup [4] '(VALUE . date_sold))
4. (Find '(RELATIONSHIP . <) [5] '(VALUE . 100k))
5. (Lookup '(OBJECT . NIL) '(VALUE . selling_price))
```

Task processing starts with the *compute* task and proceeds until the *lookup selling_price* task. During the task processing phase, the designer decides what data to pre-process, what data to show to users, and how to constrain the mapping from data to graphics in order to facilitate perceptual processing. Once all the embedded tasks are processed, the search algorithm proceeds to the data attribute mapping phase, where all data attributes deemed necessary in the task processing phase are mapped to graphical properties. The mapping decisions are subject to the perceptual constraints placed during task processing.

Branching in the search procedure occurs when there are alternative methods for achieving the same goal. In Figure V-5, a black circle indicates these *branching* or *alternative* points. An important branching point, for example, occurs after the "*Process next task*" node. One alternative is to process the task with data transforms (i.e. have the system pre-compute the task results). Another alternative is to map the task data to graphical properties and let users derive the task results perceptually. Each of these alternatives causes a new path to be created in the search tree. To further differentiate these "*branching points*" in Figure V-5, we curve the arrows originating from them while leaving all other arrows rectangular. In the next sections we describe the two search phases, *task processing* and *data attribute mapping*. We will show how a search tree is constructed and what state information is stored within its nodes during each of these phases.

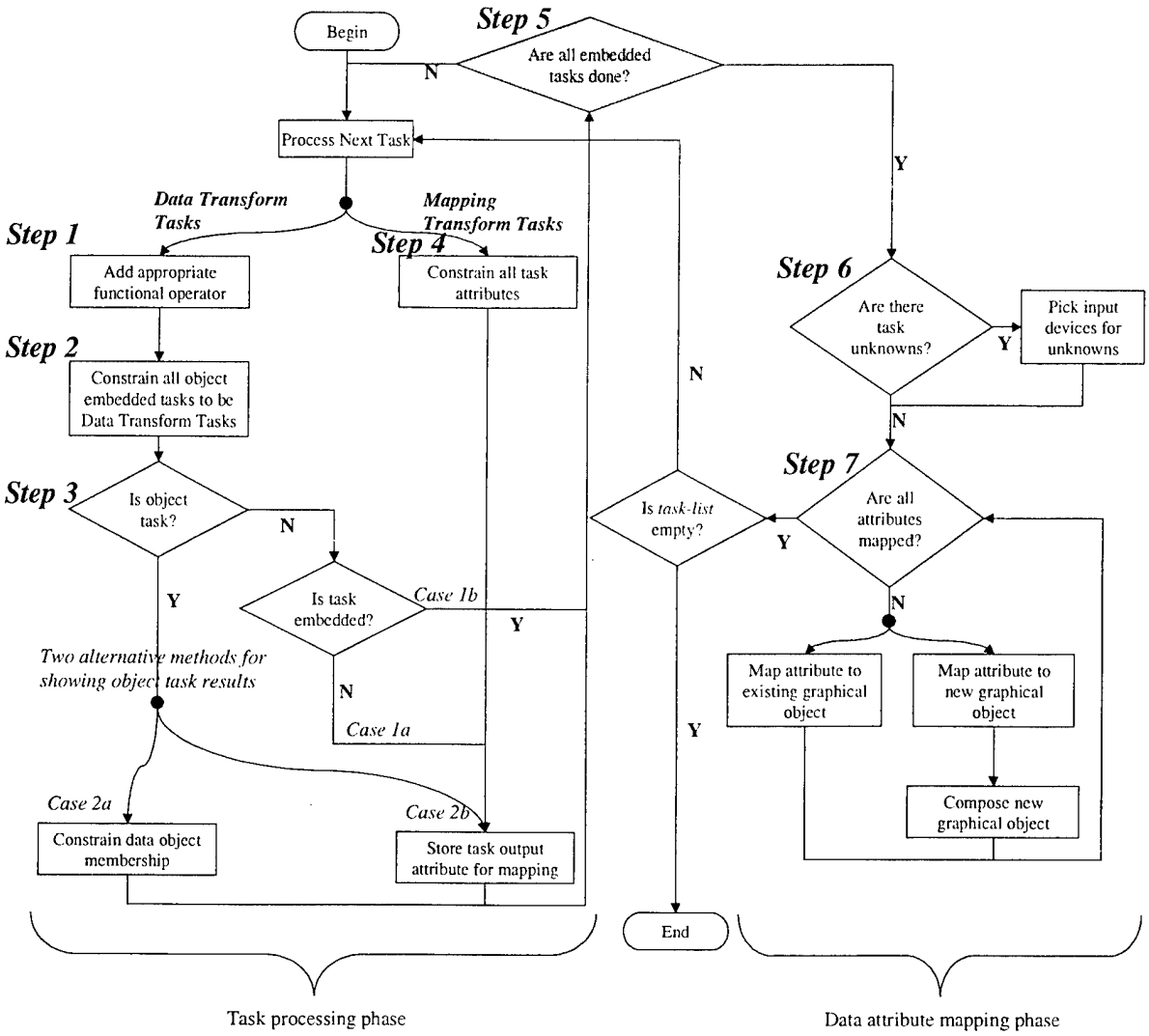


Figure V-5: Flowchart of AVID search strategy. Consists of two main phases: 1) task processing phase and 2) data attribute mapping phase

V-2.1.1 Task Processing

At the start of the search procedure the search tree contains a single root node that has a list of all outstanding tasks ordered from outermost to innermost. For example the partial search tree in Figure V-7 reflects the design space of the house example task described earlier in this section. The root node of this search tree contains a list of task structures related to the house example. Task processing begins with the first task in the root node, which is the *compute-subtract* task. Each task may be performed through data summarization and manipulation operators or by mapping data attributes of the task to appropriate graphics. As is shown in Figure V-7, each of these alternatives generates a new path in the search tree (*node-1* is the data transform alternative and *node-8* is the mapping transform alternative). New additions to the node state at each step are shown in **bold red letters**.

Note that we have cut out some nodes from the search tree in Figure V-7 because of space constraints. In particular, *node-1* and *node-8* are linked to the root node by dotted arrows to indicate that some intermediate node states along these two paths have been culled out. Both nodes show the state of the design after processing the first three tasks (i.e. the *compute* task and both of its embedded *lookup* tasks). We collapsed these two path segments because in both cases, the embedded *lookups* are constrained to the same task processing method as their parent *compute* task (i.e. both *lookups* are constrained to data transforms in *node-1* and constrained to mapping transforms in *node-8*). As a result there is no branching in the tree within these segments and collapsing them does not remove any information.

In the following paragraphs we describe the steps associated with the data and mapping transform processing alternatives (as is shown in Figure V-5) as well as the changes they make upon the node states in Figure V-7.

Alternative 1: Data Transform Processing

Step 1: Add functional operator

When we decide to perform a task through data computation (i.e. system computation), a data transform operator corresponding to the task class is generated and stored within the node. For example in *node-1* of Figure V-7 the *compute* task is performed with data transforms, thus a *BinaryCompute* data transform operator is added to the *functional-operator-list* slot of the node. Similarly each of the other data transform tasks also adds a data transform operator to the node state. These transform functions are later connected and used to create a functional design for processing the data contained within the visualization.

There is currently a one-to-one correspondence between our task classes and the data transform primitives described in chapters II and III. Figure V-6 shows all our task classes and their corresponding data transform operators.

Task operator	***Corresponding data transform primitive***
Lookup	GetAttributeValue
Compute (*mean, min, max*)	UnaryCompute
Compute (*add, subtract, divide, multiply*)	BinaryCompute
Compare	GetValueRelation
Find	Threshold
And	SetOperation

Figure V-6: Task operators and their corresponding visualization functions

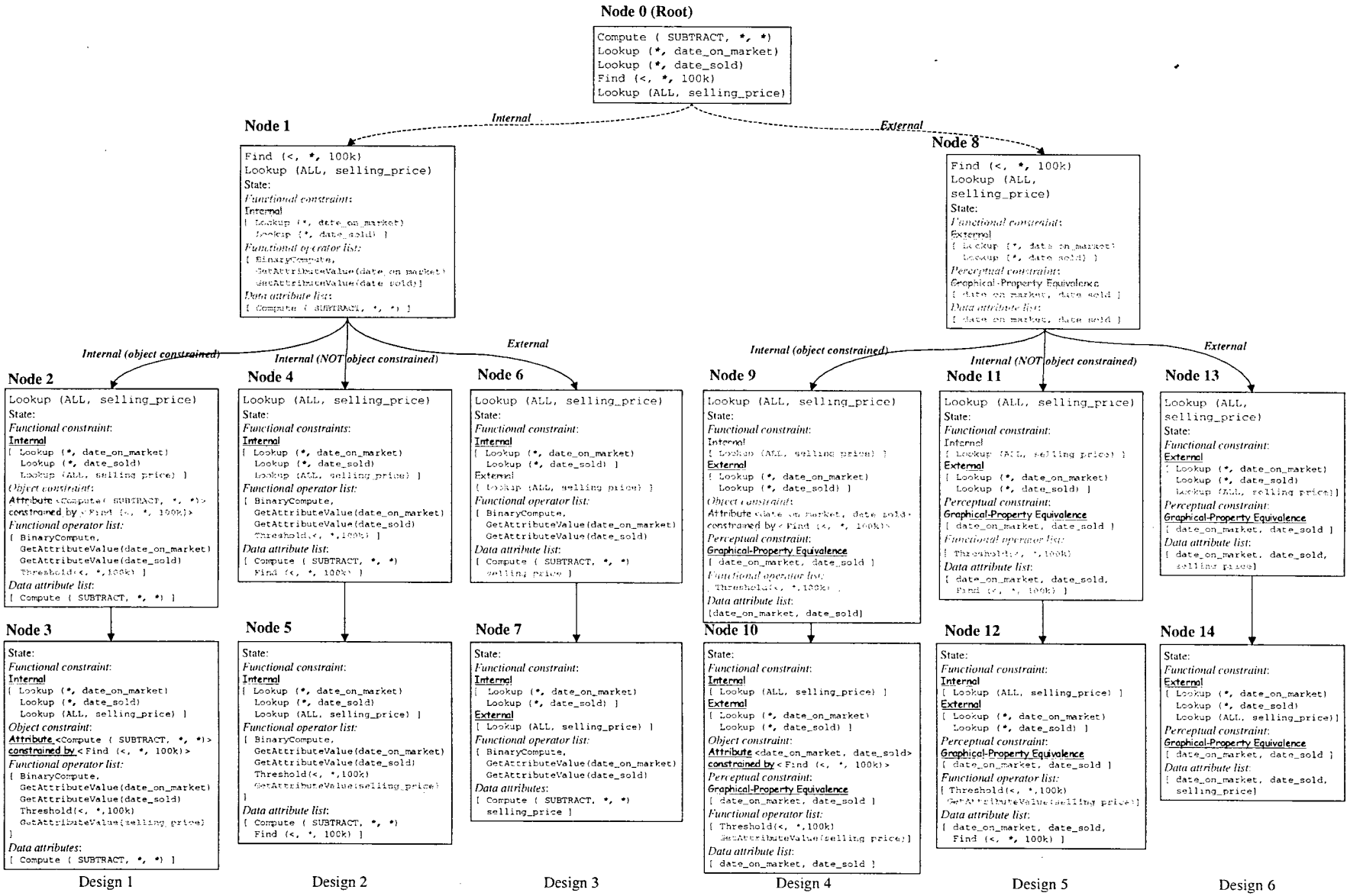
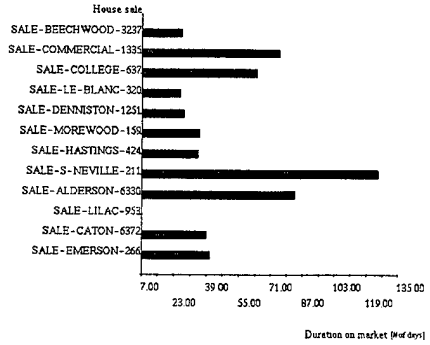
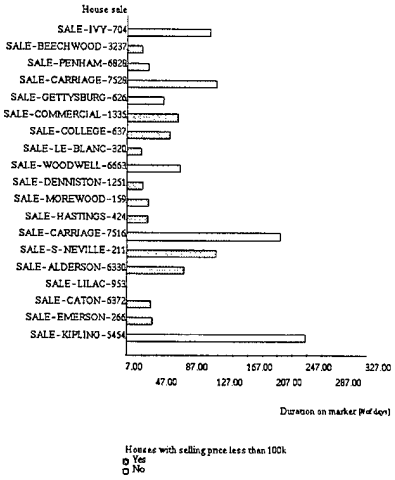


Figure V-7: Partial search tree of house example task



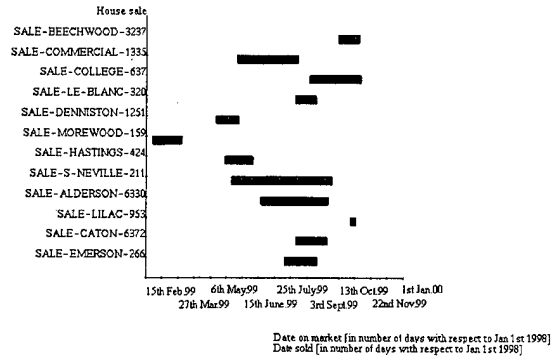
Design 1: Pure data transform design with object filtering (node-3). Duration on market is computed and mapped to the *x-axis* and only houses costing less than 100k are shown



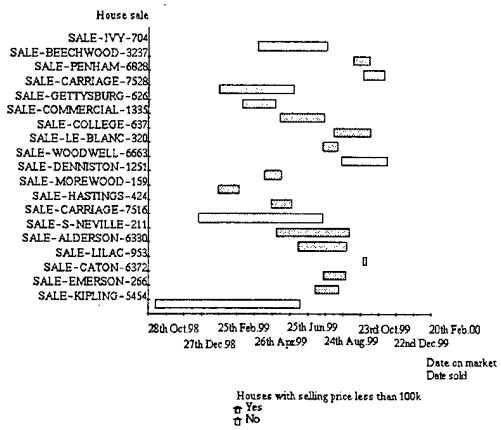
Design 2: Pure data transform design with NO object filtering (node-5). Duration on market is computed and mapped to the *x-axis*. Houses costing less then 100k are computed and shown in *red*.



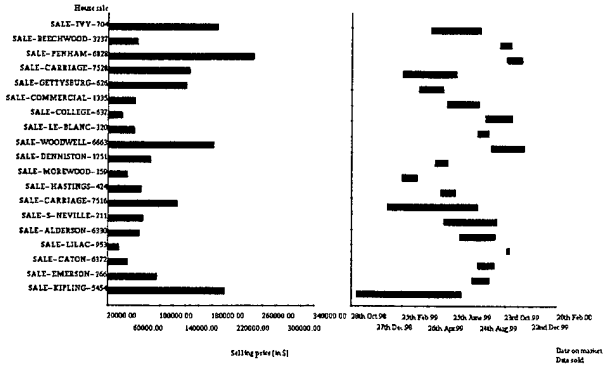
Design 3: Hybrid design with duration on market computed and mapped to the *x-axis* of the left chart. The *find* selling-price task however is performed perceptually and *selling-price* is shown on the *x-axis* in the right chart. (node-7)



Design 4: Hybrid design with *date_on_market* and *date_sold* mapped to the *x-axis* so that the duration on market can be determined perceptually. The *find* selling-price task is computed with data transforms and only those houses costing less than 100k are shown (node-10)



Design 5: Hybrid design with *date_on_market* and *date_sold* mapped to the *x-axis* so that the duration on market can be determined perceptually. The *find* selling-price task is computed with data transforms and shown using *hue* (node-12)



Design 6: Pure mapping transform design, i.e. all tasks are mapped to graphics. *Selling-price* is mapped to the *x-axis* of the left chart and *date_on_market* and *date_sold* are mapped to the *x-axis* of the right chart (node-14)

Figure V-8: Example designs generated corresponding to the 6 terminal nodes in the search tree in Figure V-7

Step 2: Constrain embedded tasks based on whether they have an object or non-object output argument

In this step we determine whether the tasks embedded within the current task are *non-object* or *object* tasks. In the first case, a data transform task constrains all of its **embedded non-object tasks** (i.e. *lookup*, *compute* or *compare*) to data transforms as well. For example *node-1* in Figure V-7 performs the *compute* task with a data transform. Since the two embedded *lookups* within the *compute* are value tasks (i.e. non-object) they are constrained to data transform functions as can be seen in the *functional-constraint* list of the node. This constraint is in place because to fulfill a task with data transform functions when its embedded tasks are performed through mapping transforms, users must perform those embedded tasks perceptually and than convey their results to the parent data transform task. Commonly this requires great precision in the mapping transform task and significant articulatory costs in conveying its results. For example in the house task, if we were to perform the *compute* with data computation but the embedded *lookup*'s with mapping transforms, the user must enter in two sets of values, one corresponding to the *date_on_market* lookup and another corresponding to the *date_sold* lookup, resulting in very significant articulatory costs.

In the second case, **embedded object tasks** (*find, AND*), do not have this data transform task constraint. Consider *node-6* in Figure V-7, the outer *lookup* task is performed with data transforms but its embedded *find* task can be performed with a mapping transform because it is an object task. While articulatory costs tend to be large for multi-value entry, they are significantly smaller for selecting a set of objects. Input devices such as *lassos* and *bounding-boxes* allow multiple objects to be selected simultaneously, while value entry must be performed individually, incurring a cost for each input value. *Lassos* and *bounding-boxes* are especially effective in the case where the resulting object set is dependent on the objects' positions. In addition input devices are often not even needed for embedded object tasks because users can perceptually identify those objects of interest and then just lookup their attributes. There is no need to convey to the system which objects fulfill the *find* task conditions unless we want to filter the objects within the display. For example consider *Design 3* which allows users to perceptually search for houses under 100k in price by looking at the *selling-price* bar chart to the right. Once those houses are identified, users may look up their *computed* duration on market. In this design, the *find* task is mapped to graphics, while the *compute, lookup date_on_market* and *lookup date_sold* tasks are data transform computed, however, no system input is required of the user.

*Step 3: Process **current** task based on whether it is embedded and whether it has an object or non-object output argument*

Once we have added appropriate transform operators and constrained all **non-object embedded tasks** to data transforms, we check whether the task itself is embedded and whether it is an object task. Table V-2 summarizes the different actions that get carried out based on these two conditions.

Task type	Task embedding	Action taken
Non-object	*(1a) Not-embedded*	Map to graphical property.
	(1b) Embedded	Do nothing.
Object	*(2a)*	Filter visualization objects.
	(2b)	Map to graphical property.

Table V-2: Summary of actions taken based on task output and embedding status

First we consider non-object tasks (*case 1*). For non-object, non-embedded tasks (*case 1a*), (like the *compute* task in the house example above) we assume that there is an implicit outer *lookup* task around it. This is because in data analysis we perform tasks to gain insight from their results and not for the exercise of performing the task itself. Thus for a non-embedded task, we need to map its resulting derived attribute to a graphical property. In *node-1*, for example, we add the derived attribute for the *compute-subtract* task to the *data-attribute-list* of the node for future mapping. In *design 1*, this *compute* attribute later gets mapped to *x-position*. Embedded tasks (*case 1b*), however, pass their results onto higher level tasks for subsequent processing, thus it is less important to show their intermediate results (refer to chapter IV-3.2). For embedded, non-object tasks we take no action and proceed to the next task.

When the task being considered is an object task (*case 2*), we have two alternatives. Either we can filter the graphical region(s) so that only those objects that fulfill the object constraints of the task are shown (*case 2a*) or we map the task results (a boolean attribute) to a graphical property (*case 2b*). These two alternatives result in *node-3* and *node-5* of the search tree in Figure V-7. In *node-3* an object constraint is applied while in *node-5* the *find* task result attribute is added to the *attribute list* of the node for subsequent mapping.

Choosing the filter alternative may significantly reduce the number of objects that need to be shown and thus reduce clutter (as is shown in *Design 1*). On the other hand such a design makes it harder to maintain data context because when we alter the conditions of the *find* task the data membership of that visualization changes and causes objects to shift around to fill in empty spaces or to make new spaces for additional objects. The object filter decision also constrains data membership for a set of graphical objects, and this may preclude the results of other related tasks with different data membership requirements from being shown with the same graphical objects. As a result object filtered visualizations usually tend to be less integrative, spreading the related data attributes over more objects and possibly more regions.

The second object task alternative (*case 2b*) maps the task results to a graphical property. This does not create context maintenance and integration problems, as is the case with filtering the objects in the previous case (*case 2a*).

However, this design decision results in greater perceptual complexity because more graphical objects are shown and an additional graphical property must be used to show the results of the search.

AVID takes the costs associated with these two alternatives into account during the design process. A cost is associated with mapping the task results to a graphical property. Less integrated designs that result from different object membership requirements also incur a design cost. AVID's cost structure is described in greater detail in section V-2.3.

Alternative 2: Mapping Transform Task Processing

Step 4: Add perceptual constraints

An alternative to performing a task with data transforms is to perform the task through mapping transforms which encodes task related data attributes with graphical properties. Mapping transform tasks commonly impose perceptual constraints on the data to graphical encodings to facilitate perceptual processing. To perform a *compute-add* task with mapping transforms, for example, we constrain all input child attributes to be mapped to stackable graphical objects and properties, the *compute-subtract* task, on the other hand, constrains all data attributes to be mapped to the same graphical property so that value comparisons are facilitated. In Figure V-7, *node-8*, for example, performing the *compute-subtract* task with mapping transforms causes a "*simple graphical-property*" constraint to be added. This constraint restricts both the *date_on_market* and *date_sold* data attributes to be mapped to the same graphical property. We see in *designs 4, 5,* and *6*, that both attributes are mapped to the *x-position* graphical property and this facilitates the perceptual *subtraction* task. More details on perceptual constraints are provided in section V-2.2.

Step 5: Determine if all embedded tasks have been processed

After we have finished processing a task we remove it from the *task-list* field of a node and continue to process the next task in that list. Processing continues until all embedded tasks have been visited. Once this is done we proceed to the *data attribute mapping* phase where we consider how the data attributes collected in the *data-attribute-list* slot in the node state can be mapped to graphical objects and properties.

V-2.1.2 Data Attribute Mapping

In this phase we consider all the data attributes in the *data-attribute-list* slot of a node and explore the different ways in which these attributes can be mapped to graphical properties. Preference in mapping is given to graphical properties that allow parallel processing and integrated designs with low perceptual complexity.

Step 6: Add input devices for unknown task arguments

Before we begin mapping the data attributes we first populate the visualization design with input devices for all unknown task arguments (i.e. task arguments that do not have associated data values because the user is unsure which value(s) are most suitable for the task). Unknown arguments are specified in AVID by using a '?' symbol in

place of an object or data value set. For example if we were unsure of what *selling_price* threshold to use in the house example, we would specify the task as follows:

```
(setf set1 (Find       `(RELATIONSHIP . <)
                       (Lookup `(OBJECT . NIL) `(VALUE . selling_price))
                       `(VALUE . ?)))

(Compute        `(VALUE . SUBTRACT)
                (Lookup set1  '(VALUE . date_on_market))
                (Lookup set1  '(VALUE . date_sold)) )        :loop-type one-to-one
                                                             :accuracy nil )
```

In this newly modified task, AVID will attach input devices to those design alternatives where the *find* task is performed with data transforms so that users may change the *selling_price* filtering threshold similar to a dynamic query interface [*Ahlberg, 1994*]. Input devices need not be added when tasks are mapped to graphics because unlike the data transform case, all the data for the task is shown and users can just perceptually process the data differently based on changing task conditions.

Step 7: Map all data attributes

The attribute mapping process is similar to the mapping process used by Casner and Mackinlay. Attributes are mapped to properties of an existing graphical element or to properties of a new graphical element that is then composed with current graphical objects. There are four types of composition methods, each of which produces a new branch in the search tree: *cluster*, *double axis*, *single axis alignment* and *no composition*.

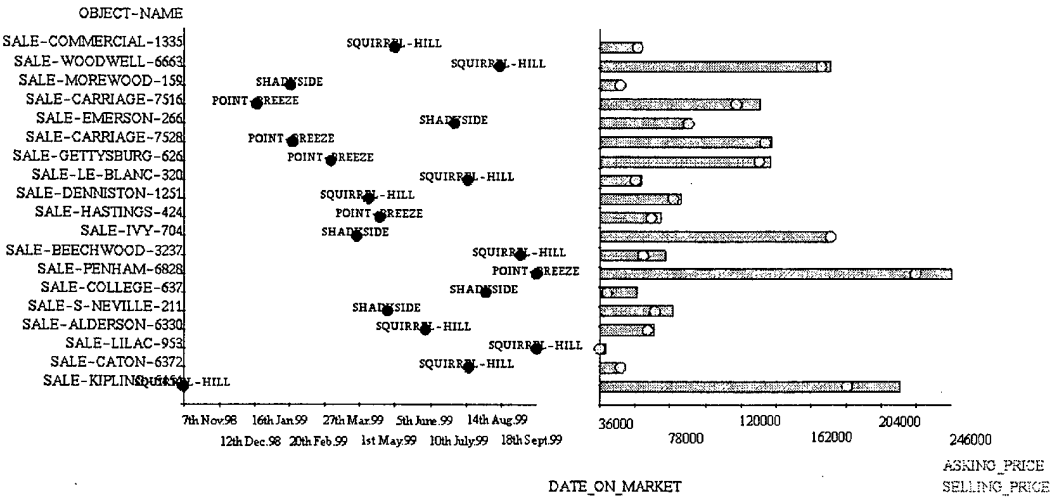


Figure V-9: Visualization design illustrating the different composition types. There is *cluster* composition in the left chart between the *labels* and the *marks*. There is *double-axis* composition in the right chart between the *marks* and the *bars*. There is *single-axis* alignment between elements in the left-chart and those in the right-chart. These composition types were first introduced by Mackinlay [*Mackinlay, 1986a, 1986b*].

Clustering ties a new object positionally to an existing object in the partial design. For example in Figure V-9, a label showing house *neighborhood* is clustered with a *mark* graphical object. *Double axis composition* adds a new

object into an existing space but does not tie it positionally to any other object within that space. For example the *mark-grapheme* and the *horizontal-bar-grapheme* in the right region of Figure V-9 are *double axis composed* because even though they reside in the same region, they do not share the same *x-position*. *Single axis alignment* adds a new object in a new region that shares at least one positional axis with an existing region. The two regions in Figure V-9 are *single axis composed*, sharing their *y-positional* axis. It is important to note that to share a positional axis both axes must have the same data type and the same min-max range. In Figure V-9, both regions are aligned on the *object-name* data type. Finally *no-composition* shows the data within a new visualization window. Different costs are associated with different graphical property mappings depending on the task and the data attribute being mapped. Costs are also different for the various object composition methods. Cost details are discussed in section V-2.3.

V-2.1.3 Post Design Processing

Once all tasks are processed and all data attributes are mapped, the visualization design is complete. During post design processing, AVID's designer performs two primary tasks:

1. *Culls out similar designs that have been generated previously*

A completed design is compared to all previous designs to determine whether there is similarity in its *structure* and *content*. If there is a structural and content match[1], then the newly completed design is culled from the design space and a new design is generated. Otherwise, the new design is transformed into our design specification language and sent to the realizer component for rendering.

2. *Transforms the visualization design into a visual and functional specification language*

Once a design is completed and determined to be unique (i.e. does not match based on structure or content to any previous design), AVID's designer translates the completed solution into a visual structure specification language and a functional specification language (developed in chapters II and III of this thesis). The translated design is then passed on to the realizer component. In appendix D-2 we describe the functional translation process, i.e., how the visualization operators within the *functional-operator-list* slot of a node are connected together and populated with sufficient information for subsequent rendering. The translation process from visual design to visual structure specification language is fairly straightforward and has been explored previously [*Mackinlay, 1986a, 1986b; Chuah 1995*], thus we will not describe this process again here.

V-2.1.4 Summary

In this section we described the search procedure used in AVID's design component. This procedure consists of two main phases: task processing phase and data attribute mapping phase. The *attribute mapping* phase was adopted from previous work on automatic systems. However, we created and added the *task processing* phase to address data transform and input device issues. Specifically we describe how to address embedded tasks and utilize

[1] Refer to appendix D-1 for details on structural and content matching.

the task embedding structure to achieve more effective data transform designs, how to filter objects within a visualization with data transforms and its impact on the graphical representation, how data transform functions are combined and attached to task parameters, and how to address unknown task arguments by attaching input devices.

During the two search phases costs are assigned to the partial visualization designs within each node of the search tree and constraints are placed on the various design elements (as is shown in Figure V-7). These costs and constraints help guide the search so that inexpressive design paths (i.e. designs that are not appropriate for the input tasks) are abandoned and promising design alternatives are explored first. Many of these cost and constraint heuristics are based on previous research that explore how visualizations can be used to amplify cognition. Card et al. [*Card, 1999*] summarizes these findings very well and we show them in Table V-3.

Increased Resources	
1. High-bandwidth hierarchical interaction	The human moving gaze system partitions limited channel capacity so that it combines high spatial resolution and wide aperture in sensing visual environments (Resnikoff, 1987).
2. Parallel perceptual processing	Some attributes of visualizations can be processed in parallel compared to text, which is aerial.
3. Offload work from cognitive to perceptual system	Some cognitive inferences done symbolically can be recoded into inferences done with simple perceptual operations (Larkin and Simon, 1987)
4. Expanded working memory	Visualizations can expand the working memory available for solving a problem (Norman, 1993)
Reduced Search	
5. Locality of processing	Visualizations group information used together reducing search (Larkin and Simon, 1987)
6. High data density	Visualizations can often represent a large amount of data in a small space (Tufte, 1983)
7. Spatially indexed addressing	By grouping data about an object, visualizations can avoid symbolic labels (Larkin and Simon, 1997)
Enhanced Recognition of Patterns	
8. Recognition instead of recall	Recognizing information generated by a visualization is easier than recalling that information by the user.
9. Abstraction and aggregation	Visualizations simplify and organize information, supplying higher centers with aggregated forms of information through abstraction and selective omission (Card, Robertson, and Mackinlay, 1991; Resnikoff, 1987)
10. Visual schemata for organization	Visually organizing data by structural relationships (e.g. by time) enhances patterns.
11. Value, relationship, trend	Visualizations can be constructed to enhance patterns at all three levels (Bertin, 1977/1981)
Perceptual Inference	
12. Visual representations make some problems obvious	Visualizations can support a large number of perceptual inferences that are extremely easy for humans (Larkin and Simon, 1987)
13. Graphical computations	Visualization can enable complex specialized graphical computations (Hutchins, 1996)
14. Perceptual Monitoring	Visualizations can allow for the monitoring of a large number of potential events if the display is organized so that these stand out by appearance or motion.
15. Manipulable Medium	Unlike static diagrams, visualizations can allow exploration of a space of parameters values and can amplify user operations.

Table V-3: How Information amplifies cognition (from Card et al.[*Card, 1999*])

In the following sections we describe the constraint and cost structures used in AVID as well as discuss their use, limitations, and how they relate to Table V-3.

V-2.2 Design Constraints

We divide AVID's design constraints into two classes according to the two primary design phases: *task processing* and *data attribute mapping*. *Task processing constraints* ensure that the visualization can be effectively controlled (small *articulatory* distance) and parsed (small *observational* distance) by the user. In addition, the visual design must be capable of expressing or processing all the information required by the input task(s) (*expressive* and *functional* distances = 0)[2].

Data attribute mapping constraints, on the other hand, ensure that the structure of the visualization is valid. For example, graphical objects within a *map* region must have their *positions* mapped to *longitude* and *latitude* while objects within a *grid* have no positional mappings. In addition, *regions* may only be aligned if they share at least one axis with the same *data type*. These *data attribute mapping constraints* follow established information design rules and their application to automatic visualization design have been explored in previous work [*Mackinlay, 1986a, 1986b; Casner, 1991; Roth, 1994*]. Therefore, in this section we will focus only on *task processing constraints*.

Task processing constraints are characterized based on three dimensions: *softness*, *scope*, and *constraint-condition*. We describe each of these dimensions next, as well as detail the primary areas in the task processing phase where these constraints get imposed.

V-2.2.1 Constraint Dimension 1: Softness

Constraints may be applied as *hard* or *soft constraints*. *Hard constraints* cannot be violated. Any search path that violates a hard constraint is considered a failure and abandoned. In AVID we use hard constraints to prevent the designer from generating visual representations that are not functionally and/or visually expressive of the input tasks.

Specifically, hard constraints are applied in AVID so that designs with positive task expressive or functional distances are never generated because they do not provide users with sufficient information to solve the input tasks. Note, however, that hard constraints are not applied to all the expressive distance measures. In chapter IV-1.2, we listed four expressiveness measures: task expressiveness, data expressiveness, correctness, and data presence. Task expressiveness and correctness are necessary conditions in any AVID generated design because they determine whether a task can even be performed and if so whether it can be performed correctly. As a result these measures are implemented as hard constraints. Data presence and data expressiveness restrictions, on the other hand, are not crucial to completing a task, thus they are implemented as soft constraints. In fact, data summarization, which

[2] Articulatory, functional, expressive, and observational distances were all described in the metrics framework in chapter IV-2.

decreases data expressiveness, can be a powerful tool for reducing graphical complexity and improving perceptual processing [*No.9, Table V-3*].

Hard constraints are also applied to prevent perceptually inexpressive designs from being generated. For example in Figure V-10, the *date_on_market* and *date_sold* attributes are mapped to two different graphical properties (*x-position* and *saturation*) making it very difficult to determine the duration on market without resorting to cognitive calculation. In this case the perceptual mappings are a hindrance because they do not enable any perceptual operators for performing the *compute* duration task (i.e. because the graphical properties used to represent the data are inexpressive, it is not possible to offload the cognitive operations onto the perceptual system [*No.3, Table V-3*]). As a result the graphical property values must first be converted back into data values and then the computation task must be performed cognitively. To prevent this, we enforce a *graphical-property* constraint (which restricts both *date_on_market* and *date_sold* to the same graphical property) as a *hard* constraint. Consider all the designs resulting from the subtree at *node-8* that has this *graphical-property* constraint (i.e. *Design 3, Design 4, Design 5*). All these designs allow the *compute* duration task to be performed perceptually by mapping both *date_on_market* and *date_sold* to the same graphical property class, namely *x-position*.

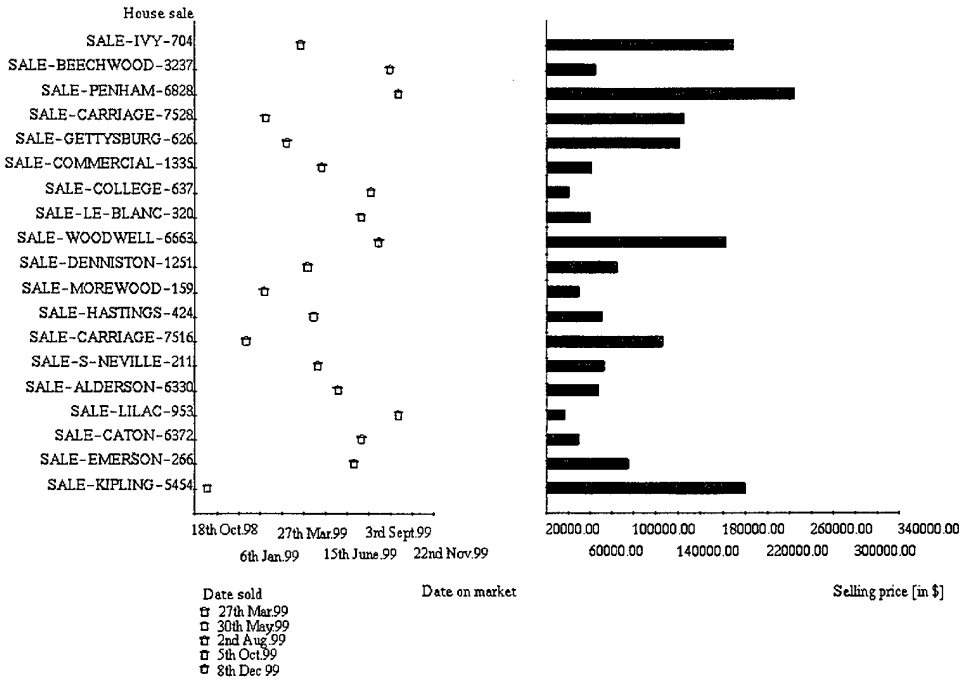
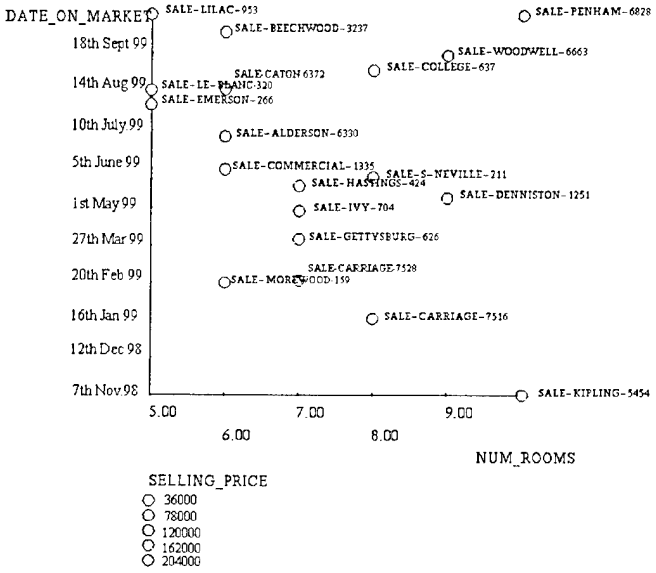


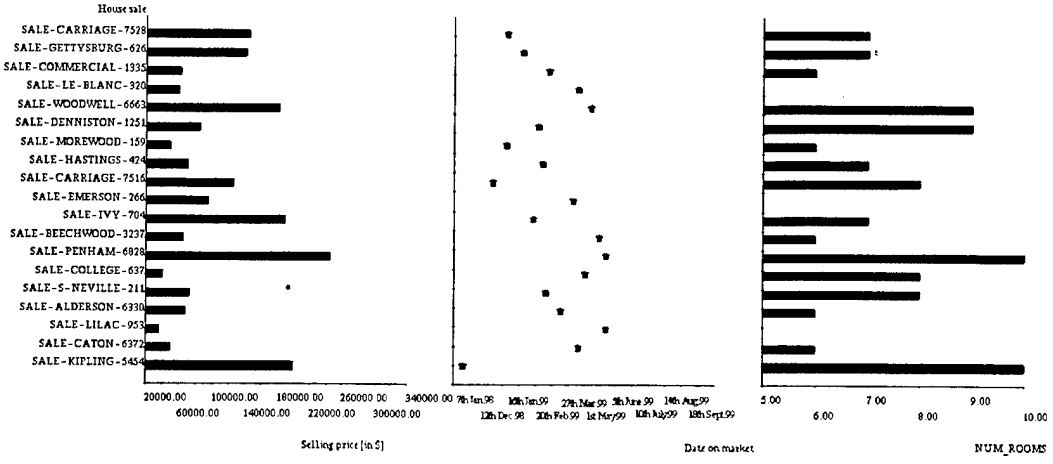
Figure V-10: Perceptually inexpressive design of the house task in Figure V-2. This is because *date_on_market* is mapped to the *x-axis* of the left chart and *date_sold* is mapped to *saturation* on the left chart. This makes it difficult to compute the duration on market because there is no perceptual operator for comparing the difference between *positional* and *saturation* values.

Soft constraints, unlike *hard constraints*, may be violated, but they incur a violation cost. Soft constraints help direct the designer to choose more effective designs (i.e. designs with smaller observational and articulatory

distances) over less effective ones. For example there are *soft* integration constraints which try to direct the designer to show all task related data within the same region [*No.5, Table V-3*]. Suppose we are searching for houses based on three properties, *selling_price, number_of_rooms* and *date_on_market*. Figure V-11 shows mapping transform solutions to the task in which all three *find* attributes are mapped to graphical properties. In Figure V-11a all the information is displayed in the same *chart* area while in Figure V-11b the information is separated over three different regions. Both visualizations can be used to solve the house search task however Figure V-11a is more effective because it requires fewer eye movements due to the more integrated design. The soft integration constraint reflects this preference.



(a) Integrated Design: *Selling_price, number_of_rooms* and *date_on_market* are all mapped onto a single region thereby facilitating the house search task.



(b) Less Integrated Design: *Selling_price, number_of_rooms* and *date_on_market* are each mapped onto a different region thereby making the house search task less efficient compared to design (a).

Figure V-11: Mapping transform designs for house search task on *selling_price, number_of_rooms* and *date_on_market*

V-2.2.2 Constraint Dimension 2: Scope

Constraints may have *local* or *global* scope. Local constraints only affect the current design decision. Once the decision is made, any related local constraints are discarded (i.e. local constraints are not propagated from one decision point to another). For example, AVID's designer has a local constraint that ensures a task can only be graphically mapped if there are appropriate graphical representations for it. Some complex tasks such as *log* or *exponent* have no appropriate mappings and thus cannot be achieved with mapping functions. This constraint only has *local* scope because once we decide that the *compute-log* task must be performed with data computation, we need not check this condition again later in the design process. I.e. the constraint only affects the current decision.

Global constraints, on the other hand, must be propagated through the design states because they may affect multiple design decisions spanning different time periods. For example the *graphical-property* constraint applied at *node-8* is a *global constraint*, and it gets propagated to all child nodes. Even though this constraint is generated in the task processing phase (i.e. during processing of the *compute*-duration task), it affects multiple mapping decisions in the data attribute mapping phase.

Before a node is processed, all global constraints stored within it are instantiated to set the current constraint context of the search path. Once we have finished processing a node (i.e. generated all of its children) we remove its constraint context and replace it with the context of the next node.

V-2.2.3 Constraint Dimension 3: Constraint Condition

Each constraint has a test condition. Inability to pass the test condition causes a violation of the constraint. This may result in the abandonment of the current design path (in the case of a *hard* constraint) or in a cost increment for the current partial design (in the case of a *soft* constraint). Constraint conditions may be placed on various elements of the visualization design or of the input task(s). The elements that may be used in a constraint differ based on the scope of the constraint. *Local* constraints can only be applied to elements and properties that are locally accessible while *global* constraints can be applied to any element.

There are three groups of constraint conditions that commonly appear in AVID: mapping constraints, task constraints, and object membership constraints.

1. *Mapping constraints*

Mapping constraints are the most common type of constraint. These constraints restrict how data attributes may be mapped onto graphical properties and graphical objects. There are two classes of mapping constraints, *simple constraints* and *complex constraints*. Simple mapping constraints are equality constraints that restrict a single aspect of a mapping to be identical with that of another. For example in *node-8* of Figure V-7 a simple *graphical-property* mapping constraint gets applied to the *date_on_market* and *date_sold* attributes. This constraint restricts the *date_on_market* and *date_sold* mappings to have identical destination graphical properties. Simple mapping constraints may also restrict a mapping property to a named constant value. For example we may constrain the

date_on_market attribute so that it is mapped to the *mark grapheme class* or to a specific region in the visualization, e.g. *region_324*.

AVID has seven different simple mapping constraints, the first five are object constraints while the last two are property constraints:

a) *Cluster constraint*: data attribute constrained to a given cluster of graphical objects.

b) *Graphical object constraint*: data attribute constrained to a given graphical object.

c) *Graphical object class constraint*: data attribute constrained to a graphical object that is of a given class (e.g. *mark*, *horizontal bar* or a *line* object class).

d) *Region constraint*: data attribute constrained to reside in a given graphical region.

e) *Region discipline constraint*: data attribute constrained to a particular region discipline (e.g. *chart, table, grid, map*).

f) *Segment constraint*: data attribute constrained to a given graphical property segment. A graphical segment links a set of data attributes to a graphical property and a graphical minimum and maximum for that segment. For two data attributes to be encoded in the same segment they must be of the same *data type* and they must share approximately equal data minimum and maximum ranges. Some example segments include *positional-axes* and *color, size* or *shape* legends.

g) *Graphical property constraint*: data attribute constrained to a particular graphical property (e.g. *position-x, position-y, shape, size, hue*, etc.).

Complex mapping constraints, unlike the simple constraints may restrict multiple graphical properties simultaneously and/or restrict the *relationships* between multiple graphical objects or properties. The primary complex constraints in AVID are the graphical property relationship constraints: *integral, conjoint* and *separable*. *Integral properties* refer to graphical properties that cannot be perceptually separated from one another. An example is *hue* and *saturation*. Both these properties determine the *color* of an object and it is difficult to perceptually separate out the *saturation* component and the *hue* component. *Integral constraints* are used to restrict the mapping of data attributes to graphical properties that have *integral* relationships. For example we may constrain the *house_neighborhood* and the *selling_price* data attributes to share *integral* graphical properties (such as *hue* and *saturation*) as in Figure V-12. This constraint facilitates our ability to search for houses with the combined attributes of low price and good neighborhood (e.g. *Shadyside*) because we only need to focus on one aspect of the graphical object, its *color* (e.g. *light pink*).

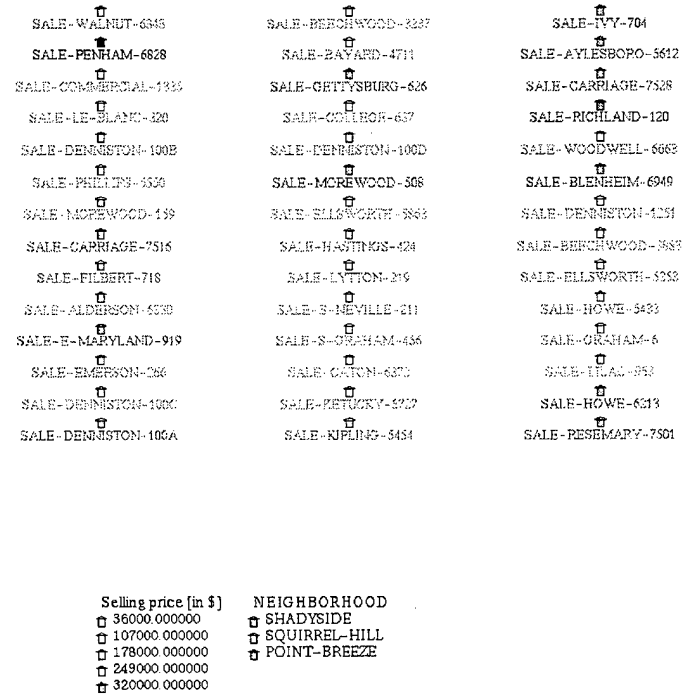


Figure V-12: Design with *neighborhood* and *selling_price* mapped to integral properties (*hue* and *saturation*). This makes combined search on both these data attributes easier because only a single emergent property (i.e. *color*) needs to be attended to.

Separable *properties* are the exact opposite of *integral* properties, in that they can only be perceived separately. An example is *size* and *hue*. Both properties do not combine to form an emergent property as in the previous case where *hue* and *saturation* combine to form *color*. As a result it is easy to view each of these properties independently. By the same token it is more difficult to perform combined property searches (e.g. find objects that are large and colored purple) compared to integral properties. Finally *conjoint properties* are both *separable* and *integral*. I.e. they can be perceived separately but at the same time they combine to form an emergent property. Examples of conjoint properties are *width* and *height*. Both properties can be perceived separately but at the same time they combine to form an emergent property namely *area*. The same is true of *x-position* and *y-position*.

Another complex constraint in AVID is the *stacking* constraint. The *stacking* constraint restricts one or more graphical objects to be positionally laid out one on top of each other. This *stacking* constraint is most commonly applied by the *add* operator to facilitate performing the summation task perceptually.

Note that mapping constraints are also special in that they get propagated from parent to child tasks. For example a simple graphical-property constraint that gets applied to the *compute-add* task shown next gets propagated to its child tasks as well so that in the pure mapping solution, all embedded data attributes (i.e. *full_compensation, full_salary, assoc_compensation, assoc_salary, asst_compensation, asst_salary*) are constrained to the same graphical property (e.g. in appendix E-2.6)

```
(Compute '(VALUE . ADD)
         (Compute '(VALUE . SUBTRACT)
                  (Lookup (OBJECT . NIL)  full_compensation)
                  (Lookup (OBJECT . NIL)  full_salary) )
         (Compute '(VALUE . SUBTRACT),
                  (Lookup (OBJECT . NIL)  assoc_compensation)
                  (Lookup (OBJECT . NIL)  assoc_salary) )
         (Compute '(VALUE . SUBTRACT),
                  (Lookup (OBJECT . NIL)  asst_compensation)
                  (Lookup (OBJECT . NIL)  asst_salary) )
```

2. *Task constraints*

Task constraints may include conditions set upon *task processing methods* (either data transform processing or mapping transform processing), *task classes* (*lookup, compute, compare, find,* AND), *task operators* (e.g. *add, subtract, multiply* or *mean*), or task input data concepts and values.

Task processing constraints are often applied to prevent bad combinations of data transform/mapping transform hybrid designs from getting generated. Step 2 of the search procedure places a task processing constraint on all embedded non-object tasks of a parent data transform task so that they are computed through data transforms as well (refer to section V-2.1.1 for details). Another interesting instance where task processing constraints are used is for imposing similarity among embedded child tasks. A good example is the *AND* task, which searches for objects fulfilling a set of data conditions. Even though it is possible to solve the task if we processed the *AND* child tasks differently (i.e. some with data transforms and some with mapping transforms as in Figure V-13) such designs are not effective.

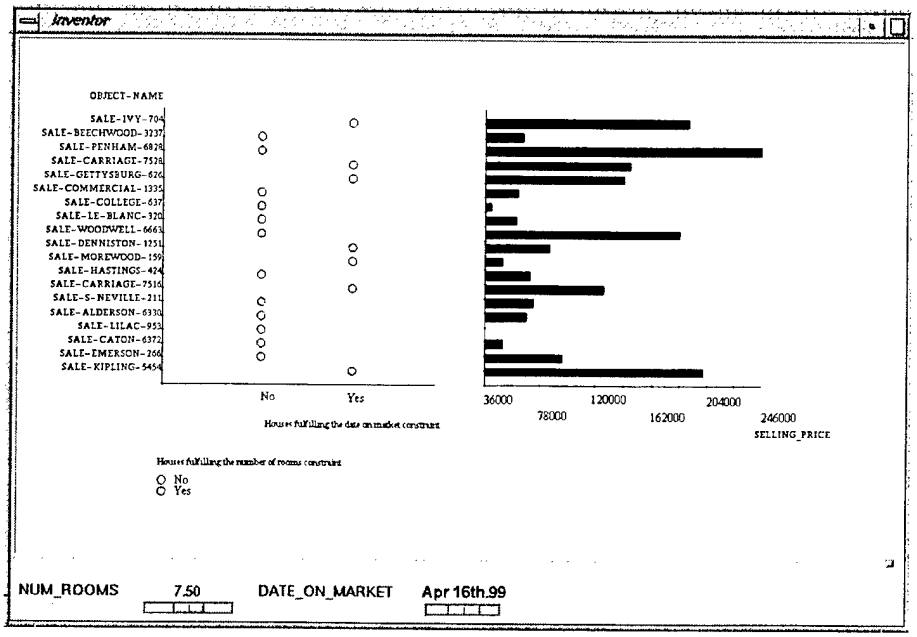
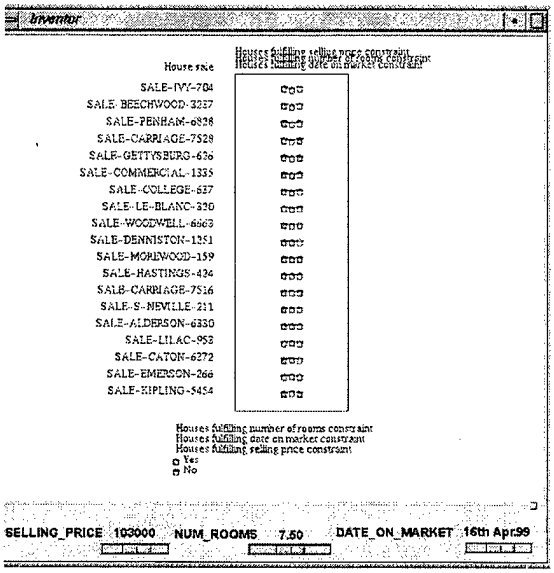
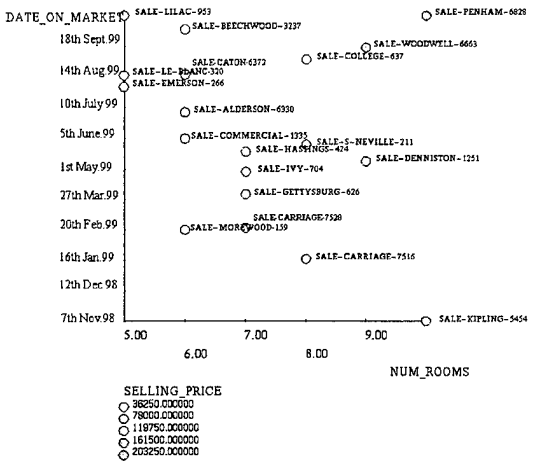


Figure V-13: Mixed task processing methods for the *AND* operator in the house search task. The *date_on_market* condition is pre-computed and mapped to the *x-axis* of the left chart, the *num_rooms* condition is pre-computed and mapped to *hue* on the left chart, however, the *selling_price* condition is performed perceptually by mapping *selling_price* to the *x-axis* on the right chart.

For example suppose we want to search for houses based on their *number_of_rooms, date_on_market,* and *selling_price* attributes. In Figure V-13 we must shift models from data computed conditions (the *number_of_rooms* and *date_on_market* conditions are system computed and mapped to *color* and *x-position*) to mapping conditions (*selling_price* is mapped to *y-position*). Pure designs such as the ones shown in Figure V-14 are much easier to understand and interpret. In Figure V-14a, each *mark* object corresponds to a search condition and whether that condition is fulfilled is indicated by its *color*. In Figure V-14b all of the raw data attributes are mapped to graphical properties. In both cases there is no confusion as to which condition is summarized (data computed) and which is not.



(a) Pure Data Transform Design. All three *AND* conditions are pre-computed and mapped to *hue* in each of the three clustered *marks*. The first mark encodes the *num_rooms* condition, the second the *date_on_market* condition, and the third the *selling_price* condition



(b) Pure Mapping Transform Design. All three *AND* conditions are represented perceptually. *Date_on_market* is mapped to the *y-axis*, *num_rooms* is mapped to the *x-axis*, and *selling_price* is mapped to saturation,

Figure V-14: Using similar task processing methods for the *AND* task

Apart from constraining the task processing methods used, we may also set task property constraints on the *task class* or *operator* so that addressing a task with mapping transforms fails if there are no appropriate graphical representations for the task, as is the case with the *log* and *exponent* operators. Task property constraints may also be applied to task arguments. For example, a mapping transform task constrains all embedded *lookup* tasks to be mapped to graphics unless there are unknown arguments associated with those tasks. This is because the data transform lookup function is only useful when it is connected to a subsequent processing function (i.e. when it is embedded within a data transform parent task) or when it is used to limit or interactively change the value set we are interested in (i.e. when it is attached to an unknown argument). Otherwise having a data transform lookup function within a mapping transform parent task has no effect because the act of mapping a lookup attribute to a graphical property implicitly extracts the data values needed from the data objects within the visualization (i.e. implicitly performs the data transform lookup function). In *node-8* both *lookup date_on_market* and *lookup date_sold* tasks are

constrained to mapping transforms (in the *functional-constraint* list) because their parent compute task is mapped to graphics and the lookup tasks have no unknown input arguments.

3. Object constraints

Object constraints restrict the data object content of different regions within a visualization. Object constraints commonly get generated to encode the results of a data computed object task (e.g. *find* or *AND* task). For example, *node-2* and *node-9* perform the *find* houses task with data transforms and show the results of the *find* to users by placing an object constraint on the region containing the *compute-duration* or *date_on_market* and *date_sold* attributes so that those regions can only contain houses that cost less than 100k. It is important to note that currently AVID does not allow a region to have inconsistent object contents. For example a region cannot contain houses that cost more than 100k as well as houses that cost less than 100k. In the future, we plan to reduce the granularity of this restriction so that it only applies to graphical object sets and not to entire regions. Object constraints commonly cause designs to be less integrated, but they have the advantage of reducing visual clutter or visual density because fewer objects are shown within each region, as can be seen in *design 1* and *design 4*.

V-2.2.4 Summary

In this section we defined task processing constraints (constraints generated during the task processing phase) based on three dimensions: *softness, scope* and *constraint-condition.* Table V-4 summarizes the constraints applied by each data transform task class and Table V-5 summarizes the constraints applied by each mapping transform task class.

Task (Data Transform case)	Constraint Condition	Soft/ Hard	Scope	Description
All	*Task processing constraint*: Non-object child tasks MUST be constrained to data transforms.	hard	global	This is to avoid the high articulatory costs associated with value entries.
Lookup	*Task argument constraint*: Task MUST have unknowns or *Task processing constraint*: Task MUST be embedded within a parent data transform task.	hard	local	If task does not have unknowns or is not an intermediate operation then a data transform lookup is unnecessary.
Compute, Compare, Find, AND	None			

Table V-4: Data transform constraints for each task class

Task (Mapping transform case)	Constraint Type		Soft/ Hard	Scope	Description
All	*Mapping constraint*: Simple-region or Simple-visualization		soft	global	Preference towards integrating all data attributes related to task within the same region or visualization window. This reduces number of eye-movements required to solve the task (**locality of processing** [*No 5, Table V-3*]).
Lookup	*Task argument constraint*: Task MUST NOT have unknowns		hard	local	If task has unknowns, then mapped lookup fails because the *GetAttributeValue* function must be used to adapt the lookup results according to changes made to the task inputs. (**manipulable medium** [*No 15, Table V-3*]).
Compute	All	*Task operator constraint*: Compute operator MUST be simple	hard	local	If compute task operator does not have perceptual parallel then mapped compute fails. (**offload to perceptual system** [*No 3, Table V-3*])
	All (except ratio)	*Mapping constraint*: Simple-graphical-property	hard	global	Ensures that data attributes are mapped to the same graphical property to facilitate perceptual computation. (**offload to perceptual system** [*No 3, Table V-3*])
	Add	*Mapping constraint*: Complex-stack	soft	global	Preference for *add-compute* attributes to be mapped to stacked objects.
	Subtract	*Mapping constraint*: NOT Complex-stack	hard	global	Ensures that data attributes for *subtract-compute* are NOT mapped to stacked objects.
	Ratio	*Mapping constraint*: Simple-graphical-object	hard	global	Ensures that *ratio-compute* attributes are mapped to the same graphical object.
		Mapping constraint: Complex-conjoint	hard	global	Ratio values must be deducible from emergent conjoint property, e.g. from combined *x* and *y position*.
Compare	*Mapping constraint*: Simple-graphical-property		hard	global	Ensures that data attributes are mapped to the same graphical property to facilitate perceptual comparison.
Find	*Mapping constraint*: Simple-graphical-property		hard	global	Ensures that data attributes are mapped to the same graphical property to facilitate perceptual comparison.
AND	*Task processing constraint*: Task processing equivalence		hard	global	Ensures that all child tasks uses the SAME task processing methods (e.g. either all data transforms or all mapping transforms).

Table V-5: Mapping design constraints for each task class

Apart from the constraints shown in Table V-4 and Table V-5, that get assigned based on the *task processing method* used and the *task-class*, AVID also contains a small set of design-wide constraints that get enforced in all visualization designs. Two important instances where design wide constraints are applied include:

1. Complex-type relationships

Mapping constraints may get imposed as a result of complex-type relationships within the data [*Roth, 1990*]. For example to express an *interval* complex type relationship between *date_on_market* and *date_sold* both data attributes must be mapped to graphical properties and graphical objects that can reflect this *interval* relationship (as

in *design 4* and *design 5*). The designer in AVID has knowledge about the complex relationships that are expressible by different classes of graphical objects. Figure V-15, for example, shows the description for the *interval-bar graphical object class* within AVID's designer. Included within this description is information on the types of complex relationships that it can express. At the start of the data attribute mapping phase, the designer determines if there are any complex-type relationships in the *data-attribute-list*. If so, the designer tries to find a graphical object and graphical properties that are capable of expressing these relationships. Complex-type constraints are declared as soft constraints.

```
(make-instance
     'grapheme-class
     'name "interval-bar-grapheme"

     ;; interval bars are capable of expressing the interval complex type with its position-x1 and position-x2
     ;; graphical properties
     'complex-types
         (list (make-instance 'complex-type
                              'type 'interval
                              'required t
                              'parameter-list (position-x1 position-x2)))

     ;; x-position can only be mapped to a quantitative data attribute
     'position-x1
       (make-instance  'grapheme-class-parameter
                        'element-type 'QUANTITATIVE)
     'member-parameters'(position-x1
                         position-x2
                         position-y1
                         hue
                         saturation)
)
```

Figure V-15: Example graphical object class specification for interval bar grapheme

2. *"Objectness" constraint* (**spatially indexed addressing** [*No 7, Table V-3*])

Figure V-16 shows an example visualization that is not expressive of the house task. In this visualization the *price* and *duration* information are separated into different spaces and it is difficult to identify which *selling_price* corresponds to which *duration* data. Our designer has a hard *"objectness" constraint* that restricts all task-related information to an object cluster or constrains each task attribute to be clustered with an *object-identifying* attribute (e.g. *object-name, house-address*). For example in *design 6* the data attributes are all tied together through the common house *object-name* attribute. It is also important to notice that object identification is more easily achieved when the identifier attribute is mapped to a *positional* property (as in *design 6*), rather than to a *label* because in the latter case we need to match *text* labels and this is en expensive perceptual operation.

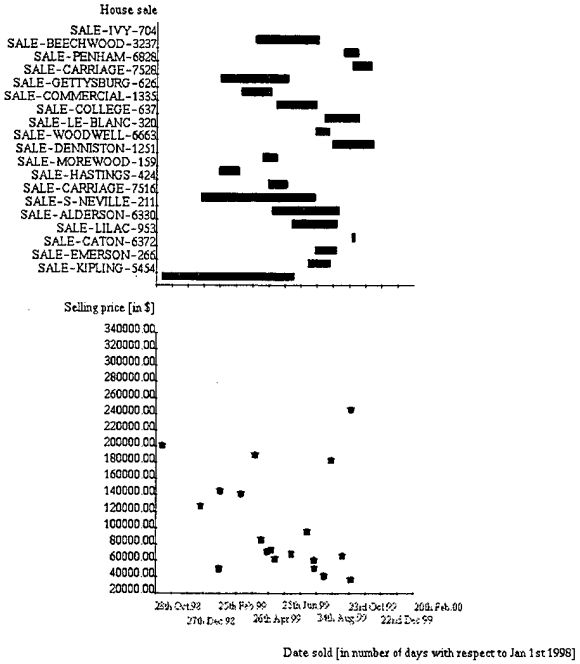


Figure V-16: Visualization design with no "objectness" constraint. I.e. it is not possible to associate which house *mark* in the bottom chart corresponds to which house *bar* in the top chart.

In summary this section describes the *task processing* and *design wide* constraints that are applied by AVID's designer. AVID also has a set of *data attribute mapping* constraints that are adapted from previous work [*Mackinlay, 1986a, 1986b; Casner, 1991; Roth, 1994*] and thus we do not describe them here. Based on the input task and data set, AVID establishes a network of constraint conditions that limit the output designs that are generated so that end users need not go through bad or inexpressive designs and can focus on a wider range of valid design alternatives.

V-2.3 Design Costs

The designer in AVID is faced with the problem of a large design space and having to present design solutions from that space to users in a timely and effective manner. In order to direct the search and generate design solutions according on their effectiveness with respect to the task we assign costs to different design decisions based on their effects to cognitive, perceptual, and articulatory complexity. The cost of each node within the search tree is the sum of its current cost and its expected future costs. The current cost of a node is the accumulated costs of the partial design at that node. The estimated cost for a node is based on the number of tasks that still needs to be processed and the number of data attributes that still needs to be mapped. Based on this cost structure AVID uses the A* search algorithm to explore the design search space.

AVID's cost structure is based on the design metrics we presented in chapter IV-1. The design metrics framework has four different goodness measures: *articulatory distance*, *functional distance*, *expressive distance*, and

observational distance. As we stated in the previous section, *functional* and *expressive* distances are primarily represented in AVID through the use of hard constraints. I.e. designs that have *functional* or *expressiveness* discrepancies with respect to the input task(s) are culled out. Thus our cost structure is primarily based on *articulatory* and *observational* distances.

V-2.3.1 Articulatory Cost Structure

Articulatory costs of a visualization interface come from the frequency and complexity of input device manipulations that need to be performed by the user. Our AVID system adds input devices to data transform tasks in two cases:

- when there are unknown arguments in the task(s), or
- when results from a mapping transform task need to be conveyed to a data transform parent task.

Input devices are added in these cases because data transform functions summarize the task data and only presents a subset of the original data values to the user. When task conditions change, the data values of interest may change as well. In order for the system to reflect these changes, users must convey the new task conditions to the system through input devices. When a task is performed with mapping transforms, however, input devices are not needed because all of the original values are displayed to the user. When task conditions change, a user perceptually filters out segments of the display that are not pertinent and only processes the relevant elements. Changing task conditions will cause a change in user focus towards different display elements, but requires no argument input.

According to the metrics framework, articulatory costs depend on the following conditions:

1. The complexity and appropriateness of each input device with respect to the task,
2. The number of input devices within the design, and
3. The number of times we expect each input device to be used.

In the following sections we describe how AVID takes each of these conditions into account in its design component.

1. *Complexity and appropriateness of input device with respect to task*

AVID attaches input devices to visualization designs so that users may change the input data values or objects to tasks. An appropriate input device is determined by analyzing properties of the data values or objects that must be conveyed and then matching that with a device that can best provide those inputs. Specifically, input devices are evaluated based on the following properties:

a) *Continuous/Non-continuous outputs*: Whether the input device is capable of expressing continuous data or graphical values. Sliders, dials, *bounding-boxes, lassos*, and *text windows* can be used to define continuous values while menus, and buttons are inherently discrete. Therefore, if we wanted to pick a reasonable *selling_price* value for the task in Figure V-2, we must use a continuous device such as a slider because *selling_price* is a continuous data attribute.

b) *Visual representation*: Whether the input device requires visual representations of particular data objects or data attributes for it to operate. For example, devices such as the *bounding-box*, *lassos*, and *mouse clicks* are applied

to visual representations of data within the visualization design. Before we can use such devices we must ensure that visual representations of the data concepts or values we want to select are available. On the other hand, devices such as the *option-button*, *scroll-list*, or *text window*, are not tied to objects within the visualization design. For example, we may select a house by entering its *address* through a text window irrespective of whether that house object is mapped to graphics. However, we can only select a house using a *bounding-box* if that house is represented and visible in the visualization.

c) *Number of objects*: The number of values, objects, or relationships that can be specified with each invocation of the device. *Bounding-boxes*, *lassos*, and *double ended sliders* for example can specify multiple objects or values. On the other hand, *text windows*, *option buttons*, and *dials* can only be used to define one value or object at any one time. Depending on the task we may only need to pick one value or one object per operation or we may need to specify sets of values or objects. For example, to enumerate all houses costing more than 100k we would need to define a set of objects thus a set input device such as a *bounding-box* would be appropriate here. On the other hand, to pick a threshold price (e.g. 100k) for filtering houses we only need to specify a single value to the system thus a *dial* or a *slider* would be sufficient here.

d) *Spatial/Non-Spatial*: Whether the input device is a spatially based device (i.e. the device defines a spatial region within the visualization window). *Bounding-boxes* and *lassos* are examples of spatial devices because they define a graphical region within the rendered visualization space. This property is important when we need to apply rendering transforms that take graphical regions as input.

Table V-6 shows all the input devices we consider in AVID as well as their status for the evaluation properties listed above. To pick an input device for a task argument, we consider the requirements of that task argument based on the properties above and match that to the input devices within AVID. We then choose a device that has all the required properties of the task argument.

	Continuity	Vis. Rep.	#-of- values	Spatial
One-ended slider	T	*	Singular	F
Two-ended slider	T	*	Plural	F
Mouse click	F	T	Singular	F
Bounding-box	T	T	Plural	T
Option Menu	F	*	Singular	F
Scroll List	F	*	*	F

Table V-6: Input devices considered in AVID with their cost properties (* indicate no constraints on an input device property)

For example, consider the house example presented earlier in this chapter. In this example, we were interested in determining the period that less expensive houses stay on the market. Suppose due to changing economic conditions, we are no longer sure what constitutes a good "expensive" house threshold. We would then alter the task

specification so that instead of setting the price threshold at 100k we use an "unknown" argument ("?"). The new task specification is as follows:

```
(setf set1 (Find '(RELATIONSHIP . <),
                 Lookup '(OBJECT . NIL), house_price),
                 '(VALUE . ?)))

(Compute '(VALUE . SUBTRACT),
        Lookup( set1, date_on_market),
        Lookup( set1, date_sold) )          :loop-type one-to-one
                                            :accuracy nil)
```

One way to fulfill this task specification is to perform the *find* task with data transforms and link the "unknown" threshold value for the *find* task to an input device. The input device properties required in this case is based on the *selling_price* data attribute, and is shown below.

	Continuity	Vis. Rep.	#-of- values	Spatial
Selling_price threshold properties	T	F	*Singular	*F

The two important constraints in this case are that the *selling_price* attribute is continuous and that the *selling_price* data attribute is not mapped to a graphical representation. The other requirements are soft requirements (indicated with an "*") because an input device that does not fulfill these conditions can still generate the types of values required for this task. AVID infers this information from the data characterization of the *selling_price* attribute and the task specification. Based on these requirements we pick the one-ended slider as the best match because it fulfills all the input device constraints including the soft requirements. Note that the two-ended slider also fulfills the two hard constraints (*continuity* and *vis.rep*), but it does not fulfill the *#-of-values* soft constraint and is therefore only a second choice.

Currently we pick the input device that fulfills all hard requirements and the greatest number of soft requirements as the best match. In AVID, we only use this "best match" input device and do not consider alternative designs that differ only in terms of the input devices used. This is because the focus of our work is not so much on choosing between multiple input device alternatives but rather on the choice of visualization functions, and how they can be used effectively in the visualization design process. By limiting our designs to only the best input device match, we limit our design search tree and increase responsiveness of our system.

2. *Number of input devices within the design*

Once an appropriate input device is chosen based on the selection process described above, a constant input device cost is added to the current design path. As a result, nodes or partial designs with a greater number of input devices will have a higher cost. This is because the more devices there are in a visualization interface, the greater the cognitive load placed upon users for understanding how to operate those devices. In addition, articulatory load is also increased because the *task specificity* is low and a greater amount of information needs to be conveyed to the

system each time we want to test out a different set of task conditions. The situation worsens when there are constraints or relationships among the different input devices, i.e. changes in one input device cause state changes to occur in other input devices. Currently, however, we do not deal with such input device relationships.

3. *Number of times we expect each input device to be used.*

The number of times input devices are used within a data analysis session depends on the number of times a user wants to vary the current task conditions. This in turn could be affected by the task operator, the data associated with the task, the importance of the task, user preferences, user experience, and the difficulty of using each input device. Dealing with many of these issues is beyond the scope of this thesis. In AVID we only focus on the extreme repetition cases. Specifically we identify tasks or task properties that will likely result in high input device use and for those instances we either abandon the design path or add a commensurate cost to the design alternative. The data transform non-object constraint described in *step 2* of the search procedure in section V-2.1.1 is an example of high articulatory cost resulting from highly repetitive input requirements.

In this section we described how input devices are chosen by AVID's design component and how costs are associated with different input device decisions. Our input device selection strategy and cost structure is simple, but sufficient to capture the design differences and illustrate the design issues we are interested in pursuing in this thesis, such as dealing with unknown task arguments, linking input devices to visualization function primitives, and capturing how articulatory costs of input devices can affect the choice of using *data transform* vs. *mapping transform* task processing strategies. A simplified input device selection strategy allows us to:

- Limit the design search procedure and simplify implementation of the system
- Focus on developing heuristics for functional operators like the ones described in chapters II and III rather than on heuristics for choosing input devices.
- Not duplicate previous work that already deals with expressiveness and effectiveness of input devices [*Card, 1990*].

V-2.3.2 Observational Cost Structure

Observational costs reflect the ease with which users can interpret a visualization interface. There are two classes of observational costs in AVID, corresponding to the two main phases of design: task processing observational costs and data attribute mapping observational costs.

V-2.3.2.1 Task processing observational cost

Task processing observational costs are accrued when tasks are mapped to graphics to account for the perceptual load placed upon users compared to data transform processing where the load is transferred to the computer system. Observational costs in this case are based on the task class, the task operator, the task input arguments, and the task properties.

1. Task class

Certain task classes are more difficult to perform perceptually than others. For example, for the task shown in Figure V-2, it is more difficult to *compute* the difference between *date_on_market* and *date_sold* than it is to *find* the houses costing more than 100k (assuming that we are using the best possible graphical properties for both tasks). Thus when we process these tasks with mapping transforms a higher cost is associated with the *compute* task compared to the *find* task. The task classes ordered according to decreasing perceptual difficulty are as follows: *Compute, Compare, AND, Find, Lookup*.

2. Task operator

The costs associated with a mapping transform task class may also vary based on the specific task operator used. This condition pertains mainly to the *compute* task that has a wide range of operators (e.g. *add, subtract, multiply, divide, mean, log, exponent*). Based on the *operator availability* heuristic in chapter IV-3.3, a higher cost is placed on compute operators that have less appropriate perceptual parallels (e.g. *mean* and *divide*) while a smaller cost is placed on those that have very effective perceptual parallels (such as *add* and *subtract*). Compute operators that have no perceptual parallels (such as *log* and *exponent*) have a hard constraint that only permits data computation.

3. Task input data

The perceptual load of a task may also depend on the task input arguments. For example *compute*, *compare* and *find* tasks are easier to perform with respect to constant input values (e.g. *find* all houses with *selling_price* greater than 100k) than with multiple data attribute value sets (e.g. *find* all universities with *avg_math_SAT* scores greater than *avg_verbal_SAT* scores). We illustrated this with the example visualizations in appendix C-4. Similarly we also showed that *enumerated* input data attributes may simplify perceptual processing of a task. This is because *enumerated* values can be represented both pre-attentively and accurately, unlike other attribute types. Thus lower task processing costs are assigned to mapping transform tasks when they have constant input arguments or enumerated input attributes.

4. Task conditions

As we described earlier in this chapter there are two task conditions: task accuracy level and task loop type.

- According to the accuracy heuristic in chapter IV-3.1, a higher cost is associated to mapping transform tasks compared to data transform tasks when *accuracy* is required. When *fuzzy* accuracy is explicitly called for, the data transform solution becomes invalid.
- According to the loop type heuristic in chapter IV-3.4, *all-to-all* tasks incur a heavy cost when we perform them with data transforms because more objects need to be shown (n^2 objects where n is the number of objects in the input data set) compared to the mapping transform alternative which only requires $2n$ objects. The cost added in this case is based on the number of data objects in the task input set.

V-2.3.2.2 Data attribute mapping observational cost

There are three classes of data attribute mapping observational costs: mapping costs, composition costs, and perceptual complexity costs.

1. Mapping cost

The mapping cost structure used in AVID is similar to the cost structures used by Mackinlay and Casner. Graphical properties are assigned costs based on how effective they are at showing different data attribute types. Preference is given to graphical properties that allow parallel perceptual processing [*No. 2, Table V-3*]. Figure V-17 shows how different graphical property classes are ordered (from most effective to least effective) based on their data attribute class (i.e. *data type*). For details on data characterization, refer to previous work by Roth [*Roth, 1990*] and Mackinlay [*Mackinlay, 1986a, 1986b*].

Data attribute class properties		**Accuracy**	**No-Accuracy**
Nominal	Enumerable	*1.* differential retinal, *2.* position, *3.* label	*1.* differential retinal, *2.* position, *3.* label
	Unenumerable	*1.* position, *2.* label	*1.* position, *2.* label
Ordinal	Enumerable	*1.* extent retinal, position, *2.* label	*1.* extent retinal, position, *2.* label
	Unenumerable	*1.* position, *2.* label, *3.* extent retinal	*1.* position, *2.* extent retinal, *3.* label
Quantitative		*1.* label, *2.* position, *3.* extent retinal	*1.* position, *2.* extent retinal, *3.* label

Figure V-17: Mapping costs ordered based on data attribute class and graphical property class

The most favored graphical property is *position* because it allows pre-attentive (parallel) perceptual processing as well as affords a relatively high degree of accuracy compared to retinal properties (such as *saturation* or *size*) which are pre-attentive but less accurate or *labels*, which are not pre-attentive. The only exceptions are in the nominal-enumerable and quantitative-accurate categories. Enumerable attributes commonly consist of only a few different values, and retinal attributes such as *hue* or *shape* can represent such attributes accurately, and pre-attentively, while requiring less display space compared to a *positional*. In the quantitative-accuracy category, *labels* are preferred because it can express the data more accurately than positionals, especially when the data range represented is large.

In Figure V-17 there are two types of retinal properties, *extent retinal* describes retinal properties capable of expressing ordered values (*saturation* and *size*) and *differential retinal* describe retinal properties that can only express unordered values (*hue*, and *shape*) [Card et al.]. Of the *extent retinal* properties we prefer *saturation* over *size* because *size* may result in occlusion problems or in expanding the display space required (less information presence). Of the *differential retinal* properties, *hue* is preferred because it is easier to pre-attentively process *hue* compared to *shape*. Note that the retinal property class is not present in the nominal-unenumerable category because data attributes of this type have too many values that have to be differentiated and it is difficult for users to associate these many values with an unordered retinal such as *hue* or *shape*.

2. Composition cost

The composition costs in AVID are assigned based on spatial locality [*No. 5, Table V-3*]. Higher costs are assigned to less integrated designs such as Figure V-11b and lower costs are assigned to more integrated designs such as Figure V-11a. Integrated designs are preferred because they require less eye-movement and visual search by the user. In addition less display space is required for the visualization, allowing more data to be shown at any one time. This is especially important for larger data sets. Based on this graphical integration rule, composition costs are ordered as follows from least cost to highest cost: use of existing graphical object, *cluster* composition, *double axis* composition, *single axis* composition, no composition (i.e. use of separate visualization window). Details on each of these composition types were discussed previously in section V-2.1.2.

3. Perceptual complexity cost

Finally costs are also added for each new set of graphical objects used. A higher cost is applied if we add new objects to a region that already contains many objects. Adding graphical objects into a visual design increases its complexity, requiring a steeper initial processing cost to learn the design structure. In addition added graphical objects may distract the user and cause perceptual interference, making it more difficult to find task related objects or identify interesting perceptual patterns.

In this work we experimented with a cost structure that seems to order the designs in a meaningful way for the classes of tasks we are interested in. We illustrated this in the GOMS analysis described in appendix E. This cost structure is just one instance of all possible cost assignments; in the future we hope to determine the costs statistically, as in a neural net. We suspect, however, that a single cost structure may not be applicable across different problem spaces and domains. The solution may lie in identifying different classes of cost structures that perform well with particular domains and tasks or letting users manipulate different cost classes manually. It is important to stress that the contribution of this work lies not in the exact cost structure provided in the expert designer but rather in identifying important aspects of the task data, task structure, and visualization design that we should attend to while assigning costs and in developing heuristics that provide general guidelines for determining which function operators and interactive devices to use, when to use them, where to use them, and what constitutes an effective combination of data transform, mapping transform and input device primitives.

V-2.4 Summary

The designer in AVID is driven by a search procedure consisting of two primary phases: the task processing phase and the data attribute mapping phase. During these two phases, there are multiple branch points that create alternate design paths in the search space. In order to direct the designer towards promising design alternatives and away from bad designs, AVID has a constraint and cost structure.

Currently AVID uses an A* branch and bound method to explore the design search space. We have also experimented with other search methods such as DFS, HILL, and BEAM. We found that although these methods can generate designs more quickly, their design quality is significantly inferior to those generated using the branch and bound method. Depending on the complexity of the task, AVID's designer may take from minutes up to several hours to come up with a design. Linear increases in the complexity of the tasks cause an exponential increase in the search space and the generation time. Optimizing the search space is ancillary to testing the theoretical concepts in this thesis, however, so we have decided not to focus on that particular aspect of the designer.

V-3 Visualization Realizer Component

The last component of the AVID system is the visualization interface realizer. The realizer accepts a visualization design specification from the automatic design component, and based on this design, instantiates or generates an active visualization interface. Each design specification has two parts:

a) *Visual structure design specification*: This specification captures the general look or structure of the graphical components within the visualization. It contains the number of graphical objects of each type (e.g. *grapheme, region, axes, legend,* etc), their object classes (e.g. *mark, bars,* and *lines* for *grapheme* objects or *charts, tables, grids* and *maps* for *region* objects), the distribution of *grapheme* objects across various *regions*, containment relationships among the graphical objects, etc.

b) *Functional design specification*: This specification describes which data, graphical, mapping, and rendering transforms are used, how these various transforms are composed, which objects they are applied to, which input devices they are linked to, etc.

The AVID realizer is divided into two components based on the two specification types described above: the graphical object realizer and the functional realizer. The graphical object realizer accepts visual structure design specifications and converts them into graphical element renderings. The functional realizer accepts functional design specifications and converts them into visualization techniques (e.g. dynamic query sliders, painting). Input device events and virtual input devices (e.g. *scroll-lists*, *option-buttons*, *sliders*) may be attached to the visualization techniques as necessary. Every visualization window is divided into two sections as is shown in many of the visualizations in this chapter (Figure V-13, Figure V-14). The top portion of the window contains graphical renderings of the data, which is generated by the *graphical object realizer*, and the bottom portion of the window contains all input devices that are generated by the *functional realizer*. Details on our AVID realizer are given in appendix D-2.

V-4 Conclusion

To showcase the applicability of our visualization framework and heuristics in "real" systems, we implemented an automatic design system (AVID) consisting of the three components: the *task interpreter, the automatic designer,* and the *visualization realizer*. The task interpreter accepts input tasks in LISP form and transforms them into a set of task and argument structures. Our task interpreter is able to deal with task embedding structure and special task conditions such as accuracy and task iteration, which was not taken into account in previous automatic visualization research. Task structures generated by the task interpreter are used by the automatic design component to guide its search of the visualization design space. In AVID we have expanded the visualization design search algorithm over previous systems to include a task processing phase for addressing data transform and input device decisions as well as a post processing design phase for culling out duplicate designs. The task processing phase deals with new issues that are unique to creating data transform designs such as processing embedded task structure, object filtering, addressing unknown task arguments, and pre-processing the input data set. In this search algorithm the task information is translated into a set of design constraints and costs that stop the designer from going down unpromising paths and direct the designer towards more effective design paths first. The translation of task information into design constraints and costs is based on the guidelines and metrics laid out in chapter IV of this thesis. Once the automatic design component finds an interesting unique solution to the input tasks, it transforms the design into a visual structure specification and a functional specification. These specifications are taken as input by the realizer component that translates the visual structure specification into a hierarchical scene graph of Inventor nodes and the functional specification into a set of acyclic functional networks. The Inventor scene graph is rendered onto the display using functions from the Inventor toolkit, and the functional networks are activated beginning with their source functions. During activation the C++ procedures associated with each visualization function in the network are executed on the input values of the functions. Functional networks may also get reactivated based on trigger events from input devices associated with the visualization design.

AVID and our interactive functional editor[3] are used to generate most of the visualization designs shown in this document. All visualizations not generated by our systems are annotated with their original sources. The wide range of example visualization designs generated by our systems shows the flexibility and generality of our theoretical framework and heuristics. Our ability to translate the theory into active systems indicates that our theory is relatively complete. In addition, the previous evaluation chapter shows that AVID produces practical design results that do indeed conform to cognitive, perceptual and articulatory complexity. The functional editor is also practical for manually creating and prototyping visualization techniques because it takes less time compared to using low-level code. Thus the implemented systems described in this chapter are good illustrations of the *generality, completeness* and *practicality* of our visualization framework and heuristics presented in previous chapters.

---

[3] Our interactive functional editor allows us to manually build visualization techniques by creating the node-link specification diagrams shown in chapters II and III. Details on this editor are given in appendix D-3.

With AVID we have expanded the design space of automatic visualization design systems. Previously, only mapping transformations were considered in the design process, but AVID is able to reason about many data manipulation and summarization operators as well as composite hybrid designs that use both mapping and data transformations to solve tasks. While we have expanded the functional design space from previous work, we can augment AVID further. An important area that AVID currently does not address is in integrating *graphical* and *rendering* transformation decisions into the design process. We will show in appendix F that *graphical* and *rendering* transformations can be very useful for solving *readability* issues that may arise in visualization designs and how our current system can be augmented with these graphical and rendering transforms. *Readability* refers to problems arising from constraints of the output medium and its interactions with our perceptual system that impede the optimal use of a visual design. Examples of *readability* impediments include occlusion among objects, display space that is too limited to show all the necessary design objects, or overly high ink density, producing visual interference.

Chapter VI: Conclusion & Future Work

In this thesis we extended automatic visualization design to include all four phases of the visualization creation process: data, mapping, graphical, and rendering. Together with this expansion, we also enable input devices to be added during design, thus enabling interactive visualization interfaces to be created automatically. Previous automatic systems focussed only on the use of mapping operations. We show that by including the full set of visualization functions we expand the range of designs that can be generated and this allows us to address data analysis tasks more effectively. Specifically we can offload cognitive tasks onto the computer system with data transform techniques as well as address readability issues such as occlusion, display density, dwarfing, and spatial separation. We show in appendix E (the evaluation section) that this added functionality can significantly decrease total task performance time. Such improvements in the quality and breadth of designs will enable automatic visualization systems to better communicate information to users as well as provide better assistance to designers for creating visualizations. The focus of our work is on the domain of exploratory data analysis however many of the theoretical concepts developed is applicable to visualization design in general.

VI-1 Summary & Relevancy of Work

We expand automatic design systems to include the four phases of the visualization creation process by developing three core technologies:

1. A framework of the visualization creation process,
2. Metrics and heuristics for measuring the goodness of visualization designs,
3. An automatic visualization design system (AVID) that utilizes our theoretical framework and heuristics to generate rendered visualization interfaces.

Our framework and heuristics are necessary for enhancing automatic visualization design research, however, they are also applicable for aiding human designers in creating and prototyping visualizations. Specifically they provide a structure and methodology for creating visualization techniques and systematically exploring the design space.

VI-1.1 Methods: Framework of the Visualization Creation Process

In chapters II and III we developed a framework for characterizing commonly used functions in each of the four visualization phases. We also show how these functions can be combined with each other, with input devices and with visualization elements through a set of composition rules. This framework is **flexible** so that as new techniques get developed, additional functions can be included with minimal effort. In addition by composing the new functions with existing functions, we can leverage off of previous operations to generate a wide range of new visualization techniques. The framework is also **general** in that

it encompasses a wide range of visualization techniques and can be integrated with a wide range of visual representations. Throughout this document we have presented *chart*, *map, table* and *grid* visualizations. These visualizations may contain *marks*, *bars*, *lines* or *text*. We have also presented many different visualization techniques including *dynamic query sliders*, *painting*, *aggregation*, *drag & drop*, *SDM techniques*, etc. Finally the framework is also **practical** for three primary reasons:

1. *Tailoring visualization techniques*: The framework provides designers with a visualization technique toolkit. This allows designers to easily modify and tailor visualization techniques to suit different task requirements. In addition it also simplifies the sharing and transfer of functionality across techniques.
2. *Design methodology*: Our framework divides the design of visualization techniques into two levels: functional and instantiation. At the functional level of design we populate the technique with the necessary functions to perform our desired tasks. At the instantiation level of design we expand the functional design with input devices and visualization elements and properties, thereby establishing the "look and feel" of the technique. This two tier design process allows functional decisions to be made free from hardware and aesthetic constraints so that we do not falsely constrain function based on form. It also allows us to identify functionally similar techniques that may have very different "look and feel" so that we may more accurately compare and borrow design strategies across techniques.
3. *Systematic exploration of visualization techniques design space*: Our framework also allows us to define and lay out the current explored areas in visualization technique design and identify areas that are less populated. By using this "map" we can systematically expand the visualization techniques design space by combining existing methods or by crafting new techniques in the less explored areas.

VI-1.2 Principles: Metrics & Heuristics for Measuring the Goodness of Visualization Designs

Our visualization technique framework provides a language for describing and creating visualization techniques. However, it does not tell us which techniques are the most effective or appropriate for solving our data analysis tasks. For any particular task, there are commonly many alternative techniques that can be used. Thus it is crucial that we have some way of measuring the goodness of these various alternatives and some guidelines for directing us towards the more promising design paths. Earlier work on automatic visualization design considered metrics and design rules for using mapping transforms based on data and task requirements. In our work we expand on previous work and develop metrics and guidelines for evaluating all phases of the visualization process including data, mapping, graphical, and rendering transforms.

Our metrics framework determines the effectiveness of a visualization design based on the four distances: articulatory, functional, expressive, and observational. This metrics framework is used by our automatic system to evaluation the effectiveness of possible design alternatives. In addition, these metrics can also be used by designers as general design yardsticks to help them create more effective visualizations

and avoid design mistakes. Note that it is also possible to evaluate the effectiveness of visualization designs by using more procedural methods such as GOMS [*Card, 1983*]. This was attempted by Lohse [*Lohse, 1993*] who automatically decomposed each task and visualization design pair into a set of GOMS operators (similar to the GOMS sequences shown in appendix E of this thesis). The problem with using GOMS in an automatic design system, however, is that it is time consuming and difficult to apply to partial designs. It can also be difficult to isolate from a GOMS evaluation which particular design decision resulted in the ultimate high or low cost of a visualization.

Based on our metrics framework we also develop a set of design rules or heuristics that help direct our automatic design system towards more promising design paths. It would be very time expensive and unfeasible to explore and rate **all** possible design alternatives before presenting them to users. As a result it is important that we have some guidelines to help focus our design efforts on the more promising design possibilities while culling out design spaces that are ineffective or inexpressive of our tasks. In chapter IV we presented design rules that help us determine when it might be more useful to perform a task or subtask perceptually by mapping it to graphics, and when it might be more advantageous to let the system compute the task through data transforms and only visualize the pre-computed results. In appendix F, we present additional design rules that consider readability issues such as occlusion, display density, data dwarfing, and information presence, and how graphical and rendering transforms can improve the readability of a visualization design.

VI-1.3 Systems: AVID – Automatic Visualization Interface Designer

To showcase the applicability of our visualization framework and heuristics in "real" systems, we implemented an automatic design system (AVID). AVID accepts one or more tasks as input and produces a set of visualization designs as output, ranked according to effectiveness of the designs with respect to the input task(s). Our AVID system was used to generate most of the visualization designs shown in this document. The wide range of examples generated by our system shows the flexibility and generality of our theoretical framework and heuristics. Our ability to translate the theory into active systems indicate that our theory is relatively complete. And appendix E (GOMS evaluation) shows that AVID produces practical design results that do indeed conform to cognitive, perceptual and articulatory complexity. Our evaluation results also show that our work significantly expands the design space of automatic visualization systems and enables more effective designs to be generated than was previously possible.

This design system can ultimately be integrated with a system like AutoBrief to enable higher level analysis and planning systems to automatically communicate complex information and relationships to users in the form of both text and graphics. Our work can also be integrated with editing and browsing interfaces similar to SageBrush and SageBook [*Roth, 1994*] to help provide design assistance to users so that the creation and prototyping of visualization interfaces can be performed more quickly, easily, and with more effective results.

VI-2 Scope of Work

We describe the scope of our work based on the three components described above. For each component we present its limitations and point to possible directions for future work.

VI-2.1 Limitations of the Framework

The visualization technique framework presented in this thesis covers many current visualization techniques. Specifically, it focuses on those operators that can be applied to data concepts, graphical objects, and the relationships between graphics and data. There are however several distinct areas that are not addressed, namely:

1. Complex Mapping Transforms

The mapping transforms considered were limited to mapping data concepts to graphical objects and mapping data attributes to graphical properties. We need not, however, limit ourselves to only graphical objects. In the *Worlds within Worlds* system [*Feiner, 1990*], for example, three-dimensional charts can be mapped within other three-dimensional charts. In this case data is not only mapped to the graphical objects within each chart region but also to attributes of the chart region itself. Through multiple embeddings the authors of the *World within Worlds* system were able to analyze a large multidimensional space.

2. Specialized transformation functions

The framework also does not explore specialized transformation functions in detail. An example class of specialized functions are those used for animations such as fade in/outs. Other specialized transformations not dealt with are space distortion techniques such as those used in the Fisheye lens [*Furnas, 1991*], and the Hyperbolic space [*Lamping, 1995*]. These distortion techniques have been analyzed to some degree by Leung et.al. [*Leung, 1994*]. Even though the framework does not provide a characterization of these animation and distortion techniques, they can be integrated into the system as additional transformation operators or as black-box operators, if need be. However, more work still needs be performed within each of these specialized areas (e.g. animation, space distortion) to define the types of functions that are common and useful.

3. Windowing operators

The framework also does not deal with windowing operators such as popping up windows, raising or lowering windows, or changing the size of windows. Primarily, this is because such low-level operations should or are already captured within the specification of the virtual input devices. While designing visualization techniques, we should not have to concern ourselves with these low-level interface operations.

4. *Workspace metaphors*

The framework does not deal with the issue of designing a set of consistent visualization techniques that fit within a common workspace metaphor (e.g. the *Mac* user interface metaphor or the *Windows* metaphor). This as an important and significant area of study, but is beyond the scope of this thesis.

5. *Conflict resolution*

At the end of appendix B-6.2 we began to deal with conflicts that may occur between visualization techniques. However we only considered conflicts along five different dimensions and between pairs of primitive techniques. More conflicts will be revealed with a deeper study of this issue. In addition, our framework does not deal with conflict resolution.

6. *Usage information*

We do not consider collecting or applying usage information in our visualization system. Some example usage information includes which objects were selected most, which objects were last selected, which objects were commonly grouped together, etc. Such information may be very useful for providing good defaults to users, and may also be useful for informing the designer of work practices within a domain.

7. *Scientific Visualization vs Information Visualization*

The visualization technique framework presented in this thesis only deals with information visualization. Another large area of study is scientific visualization. Scientific data usually has a strong physical correspondence and contains very spatially oriented information. Information visualizations, however, represent abstract data that do not have a physical correspondence and are not inherently spatial. It is therefore not surprising that the requirements for these two areas can be quite different. There are several commercial frameworks available for describing scientific visualization techniques with limited interactions [*Brodlie, 1991*]. In appendix B-1 we compare our framework to these other existing frameworks.

8. *Functions within functions*

All of the object definition and transformation functions considered are applied to either data, graphical or annotation objects. We do not consider functions that can be applied to other functions or that generate new functions as output. Most common visualization techniques do not require such complex functional interactions. This class of operations, however, are interesting to consider and may produce very powerful visualization techniques.

VI-2.2 Limitations of the Metrics & Heuristics

In our work we identified two areas where the functional expansion enabled by our work can have the most impact over previous systems:

1. By offloading task cognitive operations onto the computer system with data transform techniques in addition to offloading them onto the user's perceptual system using mapping transform techniques.
2. By considering readability issues and examining when and how the four visualization transformation classes can be used to solve these issues.

We therefore only consider design rules and heuristics for these two areas. Other interesting design areas that we do not consider include:

1. Design heuristics that use graphical and rendering transforms to solve tasks rather than just readability problems

In addition to solving readability issues, graphical and rendering transforms can also be used manipulate the elements within a visualization to change the general goals addressed by that visualization (i.e. to change what is expressed by that visualzation). Such techniques however tend to be difficult to use, and specialized to the task domain. Therefore, to integrate such design techniques into an automatic design system requires that we have a powerful model for understanding user knowledge and expertise as well as capturing specific domain knowledge. The system would also need to provide better instructions to coach users on how the techniques should be used. Because of these complexities we save consideration of these issues for future work. In our work we focus on considering how graphical and rendering transforms can be used to only solve readability problems.

2. Design heuristics for three and four dimensional visualizations

We mainly focus our work on generating two-dimensional visualizations. Our heuristics and framework easily carries over to higher dimensional visualizations. However, if we are to effectively design such visualizations we need additional sets of design guidelines that specifically deal with the issue of when it is more expeditious to map data to the third positional dimension or the fourth time dimension, rather than using aligned spaces or a *retinal* attribute.

3. Design heuristics that minimize learning time rather than task performance time

In our work we focus on developing heuristics that reduce total task performance time. As is shown in appendix E, the design rankings generated based on our design guidelines conform in most part to the GOMS estimated total task performance time, excluding learning time. To limit our problem space to a reasonable size, we assume that the users of our system are expert users who are familiar with all the visual and interactive design classes generated by our system. Thus learning time for each design is consistent and negligible. In the future it would be interesting and useful to expand our set of heuristics to include rules that take learning time into account as well as task performance time. For example we may want to give a better ranking to designs that have a consistent look or interactive metaphor to a previous design because then the learning time for that design would be much smaller than a totally new and different design. We

may also want to give preference to more conventional, well understood designs, because users would already know how to interpret and use them.

4. Design heuristics for advanced tasks involving data patterns and trends

Appendix C-3 describes two task classes: simple tasks that involve processing pairs of values, or more complex tasks that involve sensing gestalt patterns from a set of values, e.g. looking for data clusters, patterns or trends. The heuristics developed in this work is more focussed on simple value pair processing tasks, e.g. *lookup, find, AND, compute* and *compare*. This level of tasks is what was dealt with in previous automatic visualization research and we decided to focus on the same task classes. The focus of our work is on expanding the design space to include all the functions in the visualization creation process. Since we are taking the first steps in exploring the use of several new transform classes in automatic design we decided that it would better serve us to start with just the simpler tasks. Developing heuristics for the more complex trend tasks however would be a very challenging and interesting problem for the future. Note that heuristics for the complex tasks have no impact on our current heuristics for our simpler value pair processing tasks.

VI-2.3 Limitations of the System Implementation

With AVID we have expanded the design space of automatic visualization design systems. Previously, only mapping transformations were considered in the design process, but AVID is able to reason about many data manipulation and summarization operators as well as composite hybrid designs that use both mapping and data transformations to solve tasks. While we have expanded the functional design space from previous work, we can augment AVID further in the following areas:

1. Integrating graphical and rendering techniques

An important area that AVID currently does not address is in integrating *graphical* and *rendering* transformation decisions into the design process. Currently we only consider the use of data and mapping functions to solve tasks. We show in appendix F that *graphical* and *rendering* transformations can be very useful for solving *readability* issues that may arise in visualization designs and how our current system can be augmented with these graphical and rendering transforms. *Readability* refers to problems arising from constraints of the output medium and its interactions with our perceptual system that impede the optimal use of a visual design. Examples of *readability* impediments include occlusion among objects, display space that is too limited to show all the necessary design objects, or overly high ink density, producing visual interference.

2. Translating high level tasks to lower level task operators

We also do not deal with the issue of how higher level domain systems can translate their tasks into the task language required by AVID. Some discussion of this issue can be found in related research by Kerpedjiev et. al.[*Kerpedjiev, 1997*].

3. *Error checking*

We do very little error checking in our systems. Specifically our system does not have mechanisms for checking the syntax of the input task specifications. Error checking and reporting commonly get very involved and bring their very own set of research challenges that is beyond the scope of our work.

4. *Limited range of visualization representations*

Our system can generate common visualization representations such as *charts, maps, tables,* and *grids* as well as *marks, bars, lines* and *text.* Other common representation types that would be interesting to add in future work include *networks, pie charts* and richer glyphs that can encode many different data attributes simultaneously. Addition of specialized glyphs may require new heuristics and constraints to be added that are specific to these new representation types.

5. *Limited range of navigation techniques*

Currently, each of our visualization designs support some point of view navigation techniques including *zoom, pan* and *rotate.* In the future it would be interesting to integrate our automatic design system with a richer front-end data navigation environment such as the Visage system. Related to this issue it would be interesting to explore which visualization functions to include as default to all generated visualization designs and which functions to include on a case by case bases.

6. *Interface for specifying visualization techniques*

In the implementation chapter of this thesis (chapter V) we presented an interface for manually building visualization techniques by constructing data flow type diagrams similar to the specifications shown in chapters II and III. To bring visualization technique construction to more mainstream use however, a simpler, more intuitive interface will have to be designed that enables non-expert users to easily access the functionality provided by our visualization technique framework.

Appendix A

Appendix to Functional Visualization Techniques Framework (Chapter II)

A-1 ODT Diagrammatic Notation

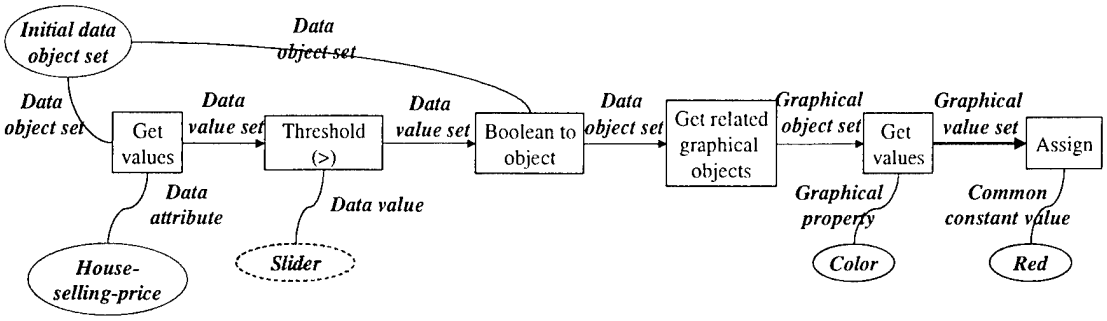


Figure A-1: Example ODT diagram for the dynamic query slider [*Ahlberg, 1992*] visualization technique

All function primitives are shown with normal *Times-Roman* font within rectangles and all inputs to the primitive functions are shown as ***italicized bold*** text within ovals. Inputs provided by users are shown with dotted ovals and those provided as designer defaults are shown with regular unbroken ovals. The directed arrows (→) connecting one primitive function to another indicate a flow of objects or values from a source function to a destination function. Arrows are sometimes also used to indicate temporal sequencing (i.e. a given primitive has to be executed before another). Temporal sequencing connections are different from regular connections because there is no information flow from the source to the destination function. Compositions can be achieved with regular connections or temporal connections depending on whether the operations have dependencies that require them to be ordered serially or whether they can be performed in parallel. In this thesis we do not differentiate between regular and temporal links because their differences do not have any significance or impact on our work.

A-2 Exploring the Space of Visualization Techniques

In chapter II-3, we outlined some simple visualization technique adaptations and expansions that can be made to current techniques to fill in less populated areas in the visualization design space. In this section we show two complete examples of how existing techniques may be combined and varied. Not all

combinations will be interesting or useful and we can identify the bad cases through common sense, general design goodness measures (chapter IV-2), or user testing.

For each of the two examples we will bring together a pair of techniques that serve different functions, and the resulting composed technique will encapsulate functionalities from both of the base methods. The first example (section A-2.1) combines: 1) the *highlight object selection* method, which allows users to select a set of objects and then colors the objects to show that they have been selected, and 2) the *dynamic query slider* method, which allows users to search for a set of objects based on specific data attributes. This example is simple and meant to illustrate how we can go about combining different functional components of existing techniques to form new behaviors.

The second example (section A-2.2) combines: 1) the *SDM distance operator* method, which improves the legibility of objects by bringing them closer together to ease comparison, and 2) the *HomeFinder* system, which allows users to search for a set of data concepts based on several constraints on their attributes. The second example explores an uncommon area in the visualization techniques' design space. As was described in chapter II-3, it is commonly the case that search techniques have simple feedback mechanisms, usually consisting of changing one graphical property in a straightforward manner (e.g. setting all the property values to a constant). In this second example we explore search techniques (e.g. *HomeFinder* system) that have more complex feedback methods, like the *SDM distance operator*.

A-2.1 Highlight Object Selection & Dynamic Query

Here we integrate the object selection and the dynamic query techniques using the composition rules described in chapter II-2. The highlight object selection technique allows users to select a set of objects through a *bounding-box* and then colors those selected objects *red*. The dynamic query technique allows users to define a set of objects by setting constraints on their data attributes. Constraints are set by using threshold functions (e.g. *greater-than, less-than, equal-to*) and the threshold value is determined by the user through a slider input device. Specifications for both these techniques are shown in Figure A-2.



Highlight object selection specification **Dynamic query slider specification**

Figure A-2: Highlight object selection and Dynamic query sliders

One way of combining the two techniques is to use pc-composition to pipe values acquired from the graphical transform of the highlight selection technique into the dynamic query technique as is shown in Figure A-3. Instead of using an *assign* graphical transform (*assign* sets the values of a data attribute or

graphical property structure but does not produce any new results) as was done in the original highlight selection technique we can use a compute graphical transform. For example, we could apply the *multiply* graphical transform to the *width* and *length* of graphical marks (to calculate their area) and then feed these values to a functional description operator as is shown in Figure A-3. This allows us to select one graphical object and subsequently make all objects that are larger than the selected object invisible. This can be an interesting method to interactively reduce occlusion in a display. Note that all functions and inputs that have been changed or removed are indicated in Figure A-3 with a red cross and new functions and inputs are highlighted in gray. The problem with culling out graphical elements based on graphical object size rather than on task related data, however, is that we may accidentally remove data elements that are crucial to our task.

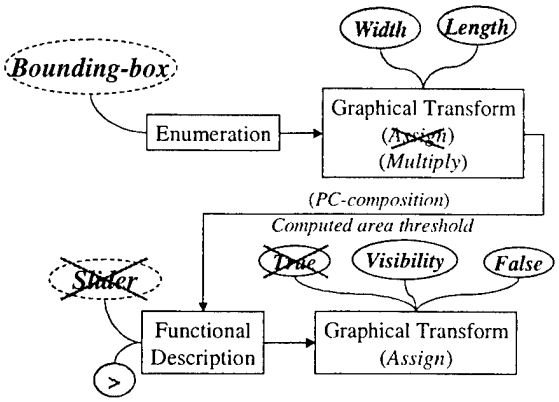


Figure A-3: P-C composition of selection and Dynamic Query

Figure A-4: Value painting specification

An alternative design applies the transform method to data values instead of graphical values. This produces a method like the value painting technique described by Eick et al.[*Eick, 1992*]. Value painting allows users to select objects in one visualization with a *bounding-box*. We then search and highlight all other objects that have the same attribute values as the chosen objects (Figure A-4). In the value painting example, we bypass the graphical transform component and simply pass on the data values as is. The *modified value painting* technique (described in chapter II-2.3) is another design alternative that can be derived from applying pc-composition to the dynamic query and selection visualization techniques.

Another way of composing the two techniques is to use object definition composition (Figure A-5). In Figure A-5 we combine both object sets from the selection technique and the dynamic query technique together with the *intersection* set-operator so that only objects that are both selected with the *bounding-box* as well as fulfill the constraint set by the threshold *slider* are highlighted *red*.

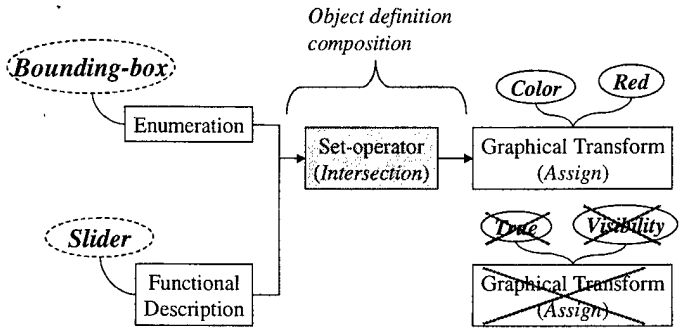


Figure A-5: Combining the object selection and dynamic query techniques using object definition composition

Another variation is to combine the graphical transform effects of both techniques with a transformation composition operator (Figure A-6). In this example, however, the two effects (*color-assign* and *visibility-assign*) do not integrate well together. This is because making non-focus objects invisible (as is done by the dynamic query slider technique) nullifies the use of the color highlighting graphical transform used in object selection. Since only focus objects are made visible, the color highlight effect is lost because all focus objects get highlighted in the same way.

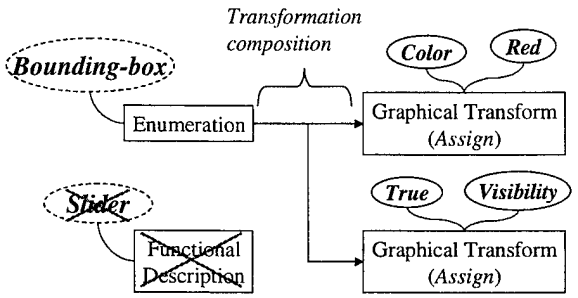


Figure A-6: Combining the object selection and dynamic query techniques using transformation composition

In this section, we see that combining the two techniques (highlight object selection and dynamic queries) allows us to integrate their different object definition and transformation methods.

A-2.2 HomeFinder System & SDM Distance Operator

In this example we explore a search technique that has more complex feedback mechanisms. Chapter II-3 showed that search techniques commonly use very simple feedback methods to show their results. For example, the HomeFinder system, the dynamic query slider technique, and the value painting technique all use simple color or visibility highlights to show the results of a search. An interesting exploration path is to see whether we could integrate a search technique with a richer graphical feedback technique that produced more interesting visual changes to objects within a visualization.

To pursue this path of exploration, we combine the HomeFinder search system with the SDM distance operator, which has rich visual feedback. The HomeFinder system was described in chapter II-2.1. This system allows users to set a number of functional description constraints on a set of data concept attributes. The graphical objects representing the data concepts are then colored based on the number of constraints passed by each concept (Figure A-7a). This is achieved through a *count* data transform that calculates for each object, the number of times it appears in a given input set. These count values are then assigned to the data concepts under consideration as a new attribute, i.e. the *count-derived-attribute*. This new attribute is mapped in a separate specification to *color* as is shown on Figure A-7b.

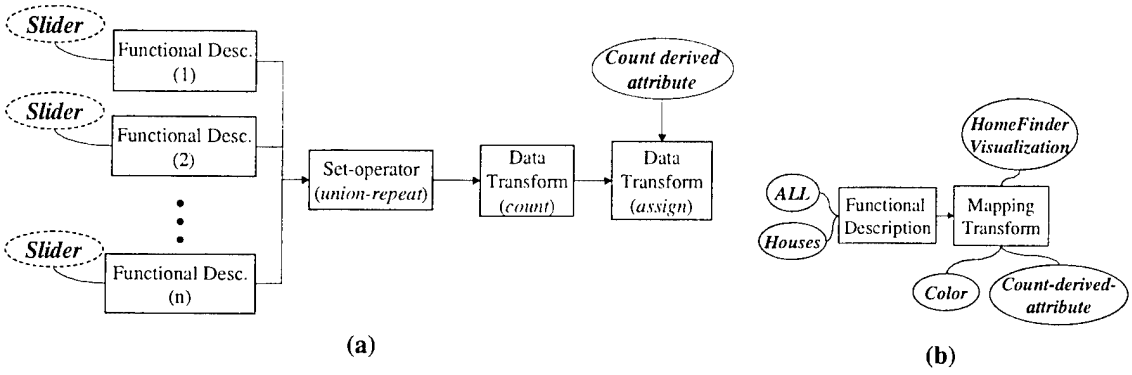


Figure A-7: HomeFinder system specification

The SDM distance operator improves the readability of a visualization by allowing users to move a set of objects to a user defined line of reference (shown in *red* in Figure A-8). By setting the line of reference to be close and orthogonal to our point of view, we improve our ability to compare object *size* or *height*, and also increase their *visibility*. This SDM distance technique is achieved by calculating for each object, the point on the reference line that is orthogonal to it (we refer to this point as the *reference point*) as is shown in Figure A-8. We then derive the distance from the original object positions to their *reference points* (i.e. *distance-to* attribute).

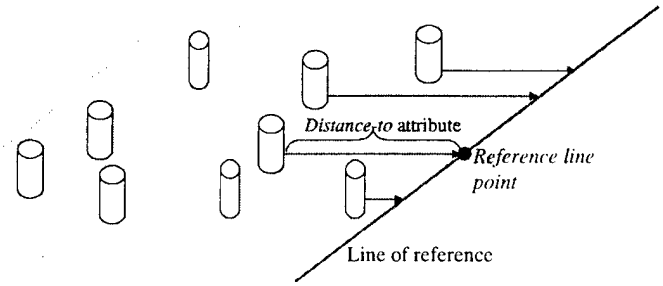


Figure A-8: SDM distance operator components: *distance-to* attribute, point of reference, and line of reference

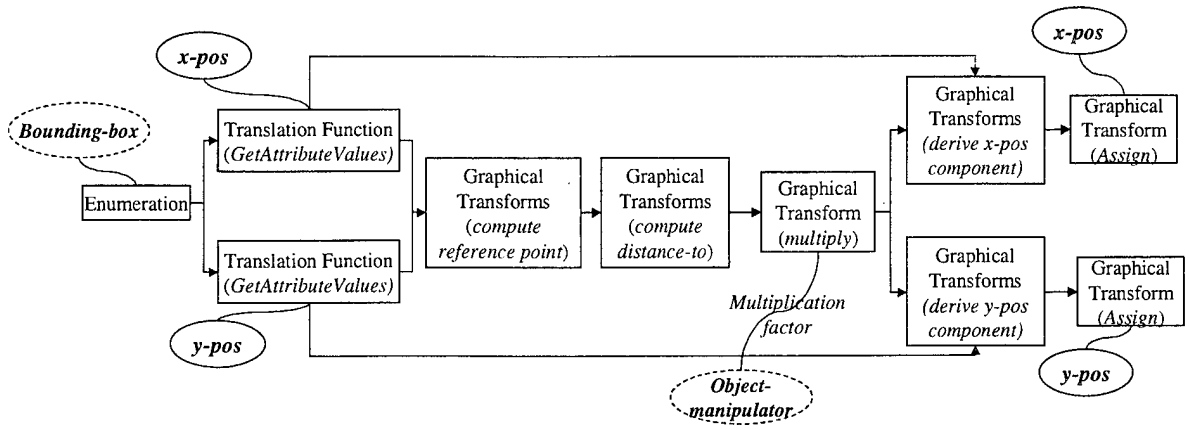


Figure A-9: SDM distance operator specification

Figure A-9 shows the design specification for the SDM distance operator[1]. We start by defining the set of objects we want to transform using the enumeration object definition operator. Multiple compute graphical transforms are used to calculate the *reference points* for these objects based on their original *xy-positions* and the position of the *line of reference*. Subsequently we compute the distances between the original object positions and their *reference points* (*distance-to* derived attribute). Users may then reposition objects anywhere along the orthogonal line between its original position and its *reference point*. Object repositioning is achieved by scaling the *distance-to* derived attribute through an *object-manipulation* input device that is tied to the *multiply* graphical transform that performs the *distance-to* scaling. Different multiplication factors cause new *distance-to* values to be computed. These *distance-to* values are then converted back into their *x* and *y* components and finally reassigned to update the objects' *x* and *y-positions*. Note that for simplicity, we represent certain sets of computation graphical transforms (*compute-reference-point, compute-distance-to, derive-x-pos-component* and *derive-y-pos-component*) with a single rectangle in Figure A-9 even though the actual operation consists of multiple simple graphical transform operators (e.g. *multiply, add, divide*).

By tying the multiplication factor to an input device, the technique allows users to slide a set of objects to and from the reference line. This enables users to maintain context of the objects' original positions. The sequence of images in Figure A-10 shows different *distance-to* multiplication factors and the virtual input device used to control those factors in the SDM system. When the *distance-to* scale is reduced to zero, all the objects get positioned along the line of reference as in Figure A-10c.

---

[1] Note that there are several *GetAttributeValue* translation functions in Figure A-9 which we have not yet described. We show these functions here however, to illustrate that the values being transformed are the *x-position* and *y-position* values of the objects. Translation functions are described in detail in chapter III-1.3, which also shows how and when such functions are added into the visualization technique specification.

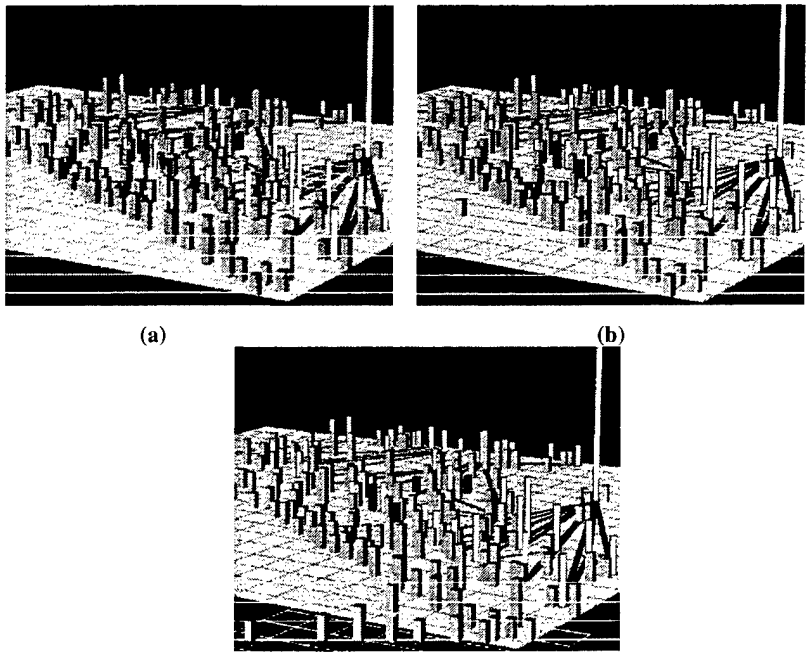


Figure A-10: Sequence of images showing different multiplication factors being applied to the distance-to attribute

By combining the HomeFinder technique and the SDM distance technique, we get an interesting synergy between their disparate goals. One way of combining these two techniques is to link the appropriateness of an object with respect to our search criteria to the distance of that object with respect to a reference element. I.e. we can interpret how good of a search match an object is by looking at its distance to a reference object. To do this we use the *count-derived-attribute* determined in the HomeFinder technique as multiplication factors for the *distance-to* computation in the *SDM* technique as in Figure A-11. Other alterations include removing the *object manipulation* input device from the SDM distance technique and applying the *distance-to* calculations to *all* the graphical objects in the HomeFinder visualization rather than just to a user enumerated set as was the case previously. In this way, the *count-derived-attribute* determines the percentage distance of every object to the *reference line*.

We can apply this technique to objects whose positions are already encoding values or to representations where the object positions are fully determined by the search results. In the former case it is important to note that the initial distance between each object and the *reference line* acts as an importance weight. Objects that are close to the reference line assign less importance to the *count* multiplication factors while objects that are farther away assign greater importance to them. For an object close to the reference line, even a high percentage change from its original distance would translate to a relatively small absolute position change. As a result objects that are far away from the *reference line* need to pass more selection constraints than objects that are closer to get to the same *distance-to* value.

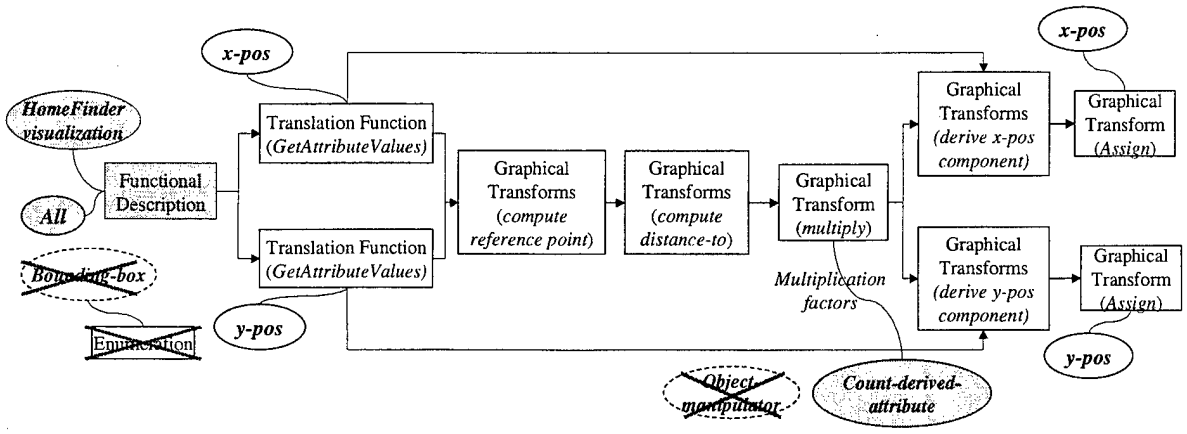
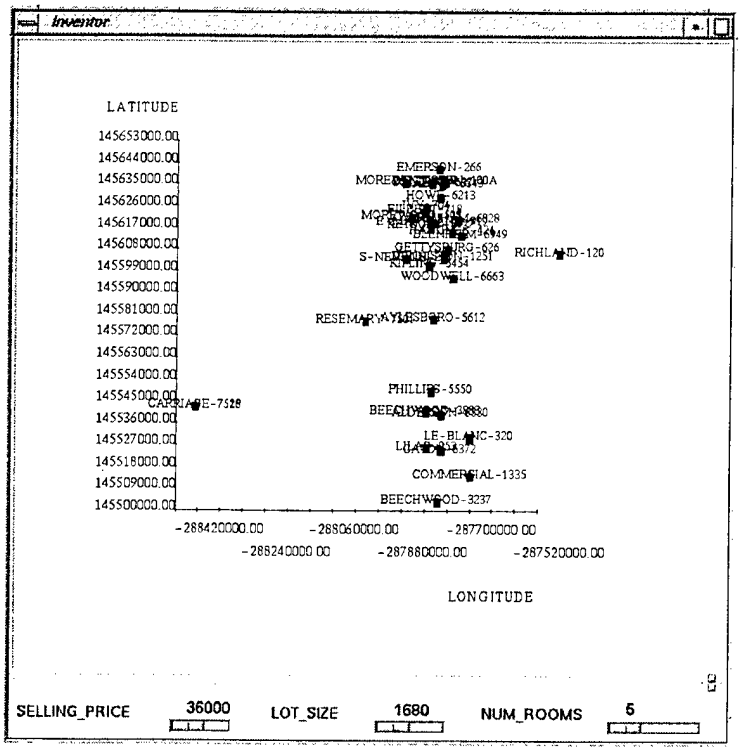


Figure A-11: Changes made to SDM and HomeFinder specifications

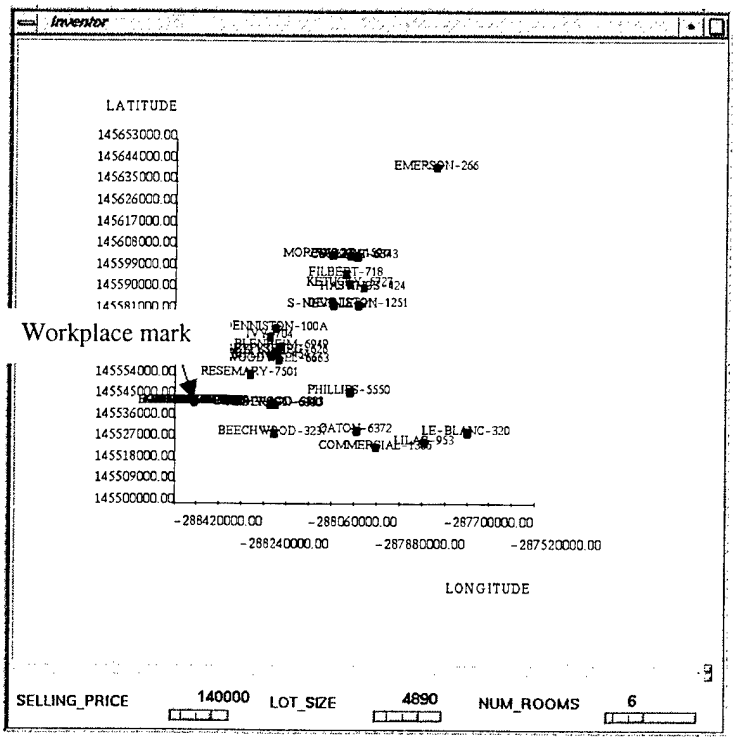
A task that is particularly appropriate for this case is one where there are natural weighting data attributes that can be easily mapped to spatial positions. For example consider the task of buying a house and suppose that we want the house to be as close to our workplace as possible. Houses that are farther away from our workplace will only be attractive if they fulfill many of our other house selection constraints such as the *num_rooms* in the house, the *selling_price*, the *crime-rate* in the surrounding area, the *availability of schools* and *hospitals*, etc. In this case, instead of using a reference line, we use a reference point, situated at our workplace (Figure A-12).

The *x* and *y* positions of the marks within the map in Figure A-12 are used to encode the *longitude* and *latitude* position of the houses that we might be interested in buying. This interface also has a set of sliders that allow us to set different house selection constraints (i.e. *greater-than selling_price, greater-than lot_size, greater-than num_rooms*) which will in turn change the percentage distance-to value of house concepts to our workplace mark. We are ultimately interested in those houses that appear closest spatially to the *red* mark. These are either the houses that are geographic neighbors of our workplace, or the houses that are farther away geographically but fulfill many of our other house selection constraints. In Figure A-12a, no workplace has been chosen, thus the position of the houses are their *longitude* and *latitude* positions.

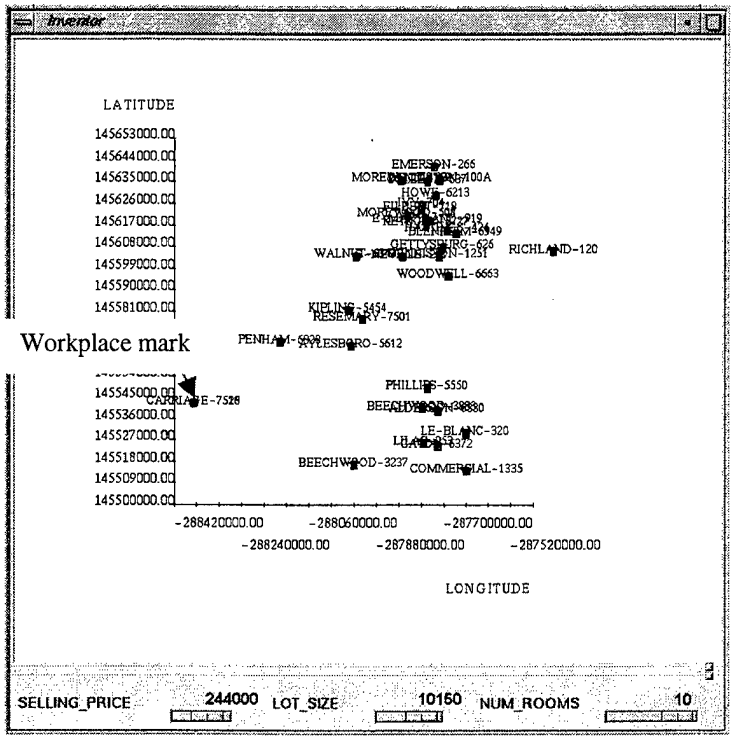
In Figure A-12b, a workplace has been selected and many of the house concepts gravitate significantly towards the *red* mark because the constraint conditions are less stringent (i.e. lower thresholds) and as a result most data concepts pass a significant number of the constraint conditions. In Figure A-12c, the threshold constraints are set higher and as the result the houses gravitate less towards the workplace mark.



No workplace chosen. Positions represent actual *latitude* and *longitude* values



(b) Low thresholds, greater proximity to workplace mark



(c) High thresholds, less proximity to workplace mark
Figure A-12: Example house selection technique

A problem with this new integrated technique, however, is that when a search is applied, the data encoded in the *x-position* and *y-position* properties is no longer valid. I.e., the positions of the marks in the map no longer indicate the geographic positions of the houses. One way to alleviate this problem is to animate the movement of objects from their original positions to their new positions. This will allow users to deduce useful information from the speed at which the objects are moving as well as provide users with context information about the object origin. In particular we would look for clusters of objects that are moving at relatively the same speed or outlier objects that are moving much faster than the other objects initially around them.

We can further extend the hybrid technique described in this section by allowing users to add several lines of reference that have different constraints attached to them (Figure A-13-left). In this case, each constraint line will apply a force onto the objects and the final position of the object would be the result of all these forces. Another variation of this technique is to use "constraint points" ((Figure A-13-right) instead of "constraint lines". This would simplify the visual representation to some degree and allow us to put more constraints into the display. When we want to put in multiple constraint points it becomes very important *where* we place these points so that we can derive useful information from the resulting object positions. For example Figure A-14 shows an effective placement of three constraint points. Objects that are in the middle of the display are the ones that pass all three constraints, objects that are on the "in-

between" points (indicated by 2's in Figure A-14) pass two of the three constraints, and objects that are at the vertices pass only one constraint. As it turns out, the display shown in Figure A-14 is very similar to the InfoCrystal system [*Spoerri, 1993*]; however, we arrived at the same design from a very different starting-point.

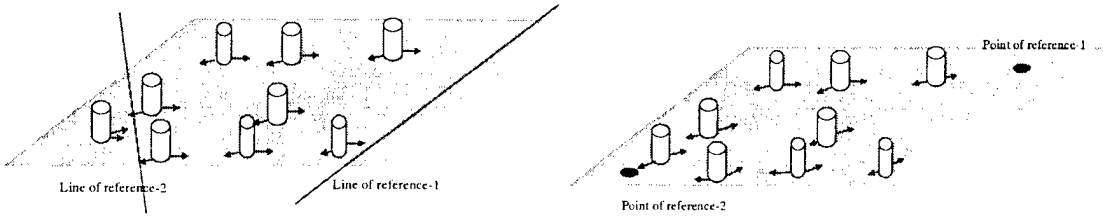


Figure A-13: Extended HomeFinder + SDM distance technique

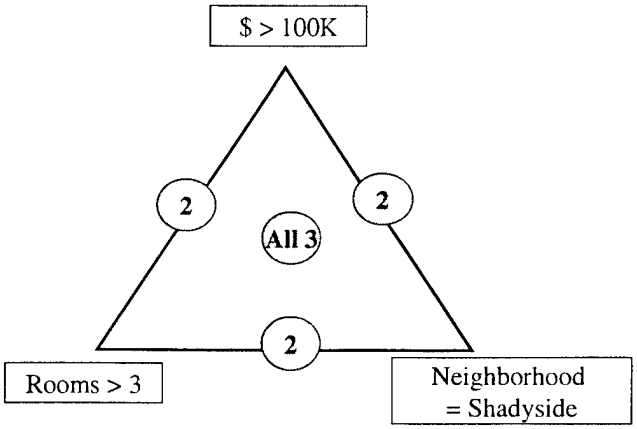


Figure A-14: Effective way of placing 3 constraint points

This section presents two detailed examples of how we can combine existing visualization techniques to form new and sometimes surprisingly novel behaviors that push the envelope of visualization technique design. Even in those cases where the combined results do not appear to have any clear use, we learn the strengths and weaknesses (new technique classes) and improve our ability to design future techniques.

A-3 Control Functions

One class of visualization functions that we did not consider in this chapter is *control functions*. Control functions regulate the flow of execution within a visualization method so that we may easily repeat operators, or choose between multiple different alternative functions. We did not include them in our description because the current visualization techniques we considered and the initial techniques we plan to automatically build with our design system do not require such functions. Future expansion of our design system however will profit significantly from the use of control functions. In this section we discuss some useful control functions and show how they may be integrated into our framework.

Sometimes it is necessary to repeat a set of object definition and transformation functions several times. Rather than having to declare the same specification over and over again, we can use *control operators*, to regulate the flow of execution of the function set. For example suppose we wanted to divide up a set of house concepts into three groups based on house price, and then color each set differently. In order to do this we could use the specification in Figure A-15 to divide up the concepts into houses that cost: 1) >= 100k and < 200k, 2) >= 200k and < 300k, and 3) >= 300k and < 400k. This can be achieved by repeating a pair of functional description functions and a color graphical transform three times; once for each house set.

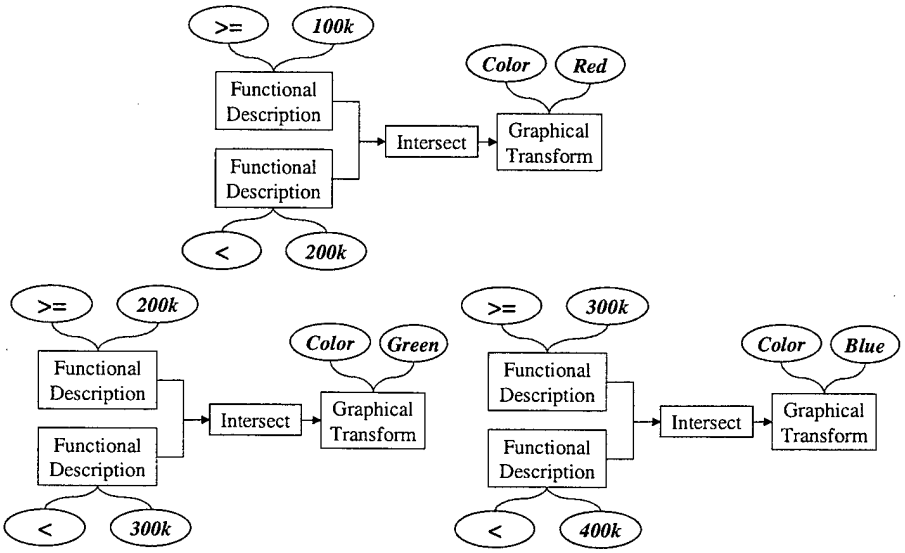


Figure A-15: Dividing up house concepts into 3 groups based on *selling_price*

A more efficient way to specify this task is to use a control operator to repeat the functional description functions and the color graphical transform so that we only need to specify them once. In Figure A-16 we use the *foreach* control operator on sets of threshold values and color values. The *foreach operator* is used to repeat a sequence of object definition and transformation functions for each member of a given set of elements. In this case the >=, <, and color graphical transform functions are repeated three times, once for each of the input arguments provided to the functions.

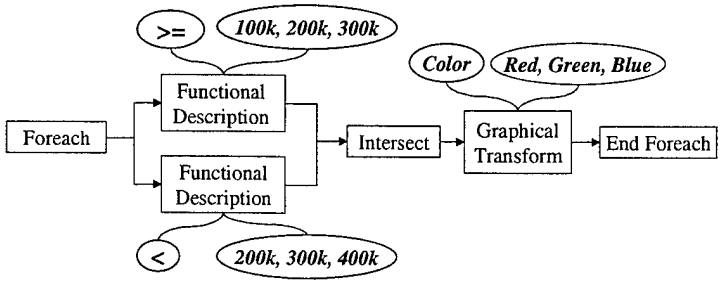


Figure A-16: Dividing up house concepts into 3 groups using the *foreach* control operator

Control operators also allow us to repeat a set of functions an indeterminate number of times, something that cannot be done through regular specification. For example, one interesting means of selecting objects is to divide up the object set into multiple partitions, where each partition contains values over particular ranges. This selection method is essentially a set of related threshold functions. We can achieve this behavior by first calculating the threshold values for each partition and then *P-C-composing*[2] that with a set of threshold functions. In the partition example shown in Figure A-17 we are calculating equi-distant partitions and then creating each partition with a pair of threshold functions. In this example the *loop* control operator is used to repeat the pair of threshold functions *n* times, where *n* is the number of partitions desired. Because we are using the *loop* control operator, we need not determine the number of partitions required during specification, and we can easily alter the number of partitions generated at any time without having to change the specification.

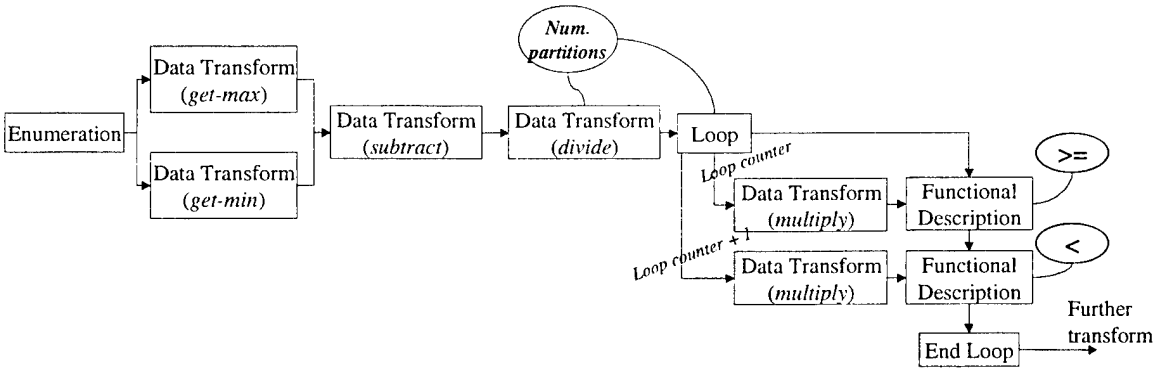


Figure A-17: Partition selection

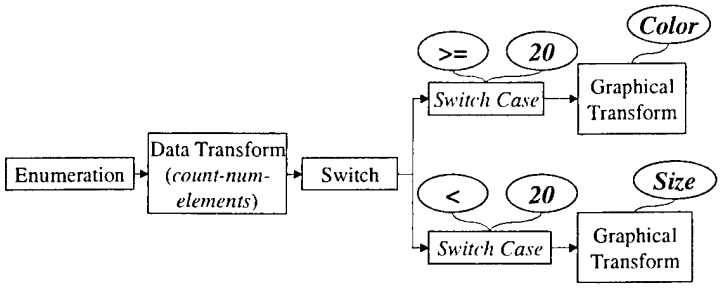


Figure A-18: Using the switch control operator to channel the execution flow

[2] *PC-composition* combines two primitive techniques by piping the outputs produced by one technique into the input slots of the other.

Finally there is the *switch* control operator that is used to choose among multiple different branches of execution based on a specified condition. For example suppose we want to highlight a set of user chosen house concepts differently depending on the number of concepts chosen. One way to do this is to use the *switch* control operator to channel the execution of the technique through different graphical transforms based on the size of the selected object set. In Figure A-18 we use the *switch* operator so that if the selected set has < 20 objects then its elements will be *color* highlighted, and if it has >= 20 objects then its elements will be enlarged (i.e. highlighted through a change in *size*).

Appendix B

Appendix to Instantiable Visualization Techniques Framework (Chapter III)

B-1 Comparison with Previous Frameworks

Part of our framework, namely the four transformation phases that correspond to the visualization generation process (Figure II-1) is very similar to Card et al's [*Card, 1999*] reference model of visualization. Both works were developed in parallel. The main difference between our visualization creation process and Card's visualization reference model is that we have an additional step of graphical transforms. This allows us to model changes in the visual structure of the visualization that is not based on any underlying data concepts, e.g. showing state information such as selection highlighting. The three classes of objects namely data, visual structure and views considered by Card also corresponds to our three realms of data, graphical scene and output media.

The four transformation classes, however, only consist of a part of our framework. In our work we define a visualization technique to have an object selection and transformation function (*ODT model*). This model is different from any other previous frameworks. Our *ODT model* is flexible because it allows us to build techniques that can create visualization interfaces from scratch or modify exiting visualizations. By including an object definition phase before transformation, we allow any type of objects to be piped into the transformation component and as a result we can build techniques that contain transformation functions that come in any order (i.e. they do not need to follow the data → mapping → graphical → rendering phases in the visualization generation process). Visualization techniques in our framework also need not contain functions from all four classes. In addition none of the previous frameworks include a compositional syntax (chapter II-2).

Another related framework from visualization techniques was presented by Tweedie [*Tweedie, 1997*]. In her framework, Tweedie described the differences in visualization techniques by using four primary criteria: data, representation, interactivity and input/output externalizations.

For example the dynamic query technique and the Table Lens technique are described in Figure B-1:

Dynamic Queries
Purpose: Find useful sets of multivariate data
Data type: Values
Representation: A scatterplot is used to display two of the attributes, the remainder are represented as sliders.
Interactivity: Data is hidden (mechanized DM) or filtered (mechanized IM) by selecting ranges on sliders.
I=O representation: Input → Output is represented

Permutation Matrices/Table Lens
Purpose: view relations in multivariate data
Data type: values
Representation: This is essentially a graphical spreadsheet (value in each cell is encoded as height)
Interactivity: Reorder the cells (mechanized DM)
I-O Representation: Only output is represented

Figure B-1: Tweedie's description of the dynamic query and Table Lens techniques

Tweedie's framework was not very appropriate for our goals in automatic design, however, because we needed a fully instantiable language of visualization techniques i.e. the descriptive language must be complete and specific enough to generate an active visualization interface. Thus unlike Tweedie's approach we needed to describe the function and structure within each technique in much greater detail. This however does not detract from our ability to analyze and categorize the various techniques as we showed in chapter II-3.

A very desirable property of our instantiable language is that it provides a common level of primitives for describing visualization techniques (as was laid out in chapter III-1). This allows us to break down high-level visualization systems and compare their capabilities at the same level of granularity. This was not true of previous frameworks [*Tweedie, 1997*], which sometimes compared visualization systems that differ in their level of granularity. For example the Table Lens and dynamic query slider techniques shown in Figure B-1 are both at two very different levels of granularity. The Table Lens system consists of multiple different technique including an attribute value sorter, a lens technique that allows users to change the size of cells, a semantic zoom technique that changes the level of detail on elements depending on their cell size, and a column move technique. In contrast the dynamic query technique is a single technique in itself. Our framework highlights such distinctions.

In our framework we have descriptions comparable to Tweedie's data, representation and interaction categories. We however, chose to separate out "goodness" measurements of input and output externalizations from our framework because this category more pertains to the effectiveness of a design or a technique rather than to describing the structure or components within a technique. Effectiveness measurements was first introduced by Mackinlay [*Mackinlay1986a, 1986b*] and in this work we extend effectiveness criteria to cover data processing and mapping designs as well as interactive methods (chapter IV).

Both Card's and Tweedie's frameworks are functional frameworks that try to capture the semantics of visualization techniques. There are also a set of instantiation languages including the Data Explorer, Iris Explorer, and AVS systems [*Brodlie, 1991*]. These systems provide an instantiable syntax and allow designers to create visualization interfaces by building data flow diagrams that convert sets of data values into visualization renderings and interface components on the computer system. These systems were also not sufficient for the goals of this thesis for several reasons:

1) They are based on scientific visualizations.
2) They are based on the use of data flow diagrams are used to create visualization systems from scratch. In our work we need to design techniques that generate new visualization interfaces from scratch as well as modify existing visualization designs. Techniques from the latter category cannot be described using any of the three instantiation packages (the Data Explorer, Iris Explorer, and AVS systems). For a more complete description of how our framework syntax differs from data flow diagrams refer to chapter II-4; 3).
3) They consist only of low-level primitives. In contrast our framework contains both a functional description as well as an instantiable description. In our work it is necessary to group and categorize techniques based on the higher more abstract functional level. This two-level description (functional and instantiation) allows us to modularize our designer so that it can initially only consider what functional primitives it needs to use to fulfill current goals. Specifics that may affect the effectiveness of a technique but not the core functionality can be considered later once we are sure that all the required function have been included. In addition, the category of the various current visualization techniques informs our designer what roles data, mapping, graphical and rendering transforms can play in the design process and how they may be usefully combined.

By using the lower level instantiable language we allow our automatic designer system to describe a visualization in sufficient detail so that it can ultimately render an active visualization interface. Unlike previous frameworks our visualization techniques language described technique in both the functional and instantiation levels using a compatible syntax as well as establishes a systematic process to move between the two different levels of abstractions: the functional level and the instantiation level. Previous frameworks either concentrate on one level or the other. We provide a common structure for representing both.

B-2 Data Flow Diagrams

It is important to note that even though our visualization technique specification may resemble data flow diagrams, they are not strictly data flows. Data flow diagrams are commonly used to analyze and understand complex systems consisting of multiple interacting processes. Our specification language, on the other hand, is meant to describe a single process (i.e. the visualization technique process), more like a high-level flow chart. The visualization technique process **does** interact with two other processes, namely

the user and the system designer, as is shown in Figure B-2. In Figure B-2, the user provides *task inputs* to the technique through the use of input devices. This input may cause system state to be updated and ultimately produce *feedback* for users in the form of visual change. The graphic designer also interacts with the visualization technique by providing *default values* to the functions within it. While our specification language does capture these relationships, its primary purpose is to encapsulate the functionality of a visualization technique in enough detail so that a working system may be generated from it. In contrast, data flow diagrams are used primarily to understand the flows or exchange of information among different processes.

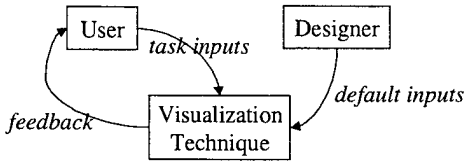


Figure B-2: Data Flow diagram showing relationships between a visualization technique, the user and the visualization designer

Other differences between our specification language and data flow diagrams include:

1. *No system state changes*: Unlike data flow diagrams we do not show system states in our specification because even though state information is important for understanding the way visualization techniques work, they are less important for capturing the functionalities that we want to perform using a technique. Including them in the specification diagram may significantly increase clutter and complexity. In any case, system state changes can be easily included in our specification diagrams as additional boxes, without changing the existing structure of the technique.

2. *Temporal links*: We have links that show flows of data from one function to another, as well as temporal links which indicate a temporal ordering between two functions (temporal links allow us to express that certain functions have to be performed before others during execution of the visualization technique). Data flow diagrams can only have data links.

B-3 Example: Generating an Instantiation Specification for Dynamic Query Sliders

This example shows how the abstract functional *dynamic query slider* [*Ahlberg, 1992*] design presented in chapter II can be augmented to form an instantiable visualization technique. Figure B-3 shows the functional specification for the dynamic query technique. The technique starts with a user-controlled, functional description, object definition function followed by a graphical transform. The first step of the

instantiation augmentation process is to determine the exact functional description function and graphical transform function to use. For the functional description we use the *threshold function* with the "*greater-than*" (>) operator. Once the objects have been defined, we need to give feedback to the user on the results of the operation. One way to do this is to attach a common identifying feature to all the selected objects. This can be achieved by using the *assign* function to set a chosen graphical property of the selected object set to a common value. Figure B-4 shows the functional specification of Figure B-3 augmented with specific instances of object definition and transformation functions.

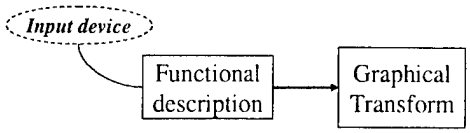


Figure B-3: Dynamic query functional specification

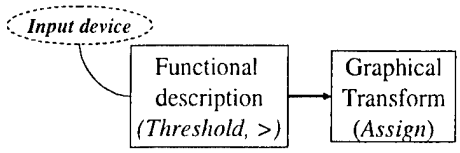


Figure B-4: Dynamic query specification with specific object-definition and transformation functions

Once the functions are chosen, we ensure that they connect correctly with one another. Sometimes translation functions must be inserted to ensure that the outputs of one function are appropriate as inputs for the following function(s). Figure B-5 shows the inputs and outputs (highlighted in gray) of the object definition and graphical transform functions that constitute the *dynamic query* technique. We start with the set of all data objects. From these objects we extract a set of data values which are fed into the *greater-than* threshold function. This function produces a set of boolean data values that are transformed into graphical values for the *assign* graphical transform function.

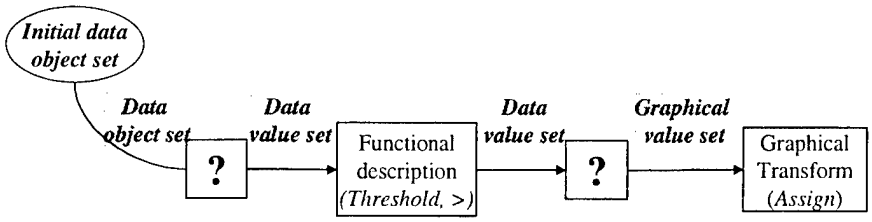


Figure B-5: Dynamic query specification with input and output types for each object definition and transformation function. The "?" boxes indicate areas where translation functions are needed to convert from one argument type to another.

Figure B-6 shows the translation functions used in this example to convert the output type of a source function to fit the input type of a destination function. The *get-values* function extracts a set of attribute values from the initial data object set. These values are fed through the threshold function, which produces a set of boolean values, indicating for each input value, whether it passed the chosen threshold. Based on these boolean values and the set of data objects considered by the *threshold* function, we identify all of the data concepts that passed the query (*boolean-to-object*). From these data concepts, we get all of the

graphical objects that are used to represent them by using the *get-related-graphical-objects* function. Finally we extract the property values that we want changed from the graphical objects. These values are passed through the *assign* function that sets them to a common constant.

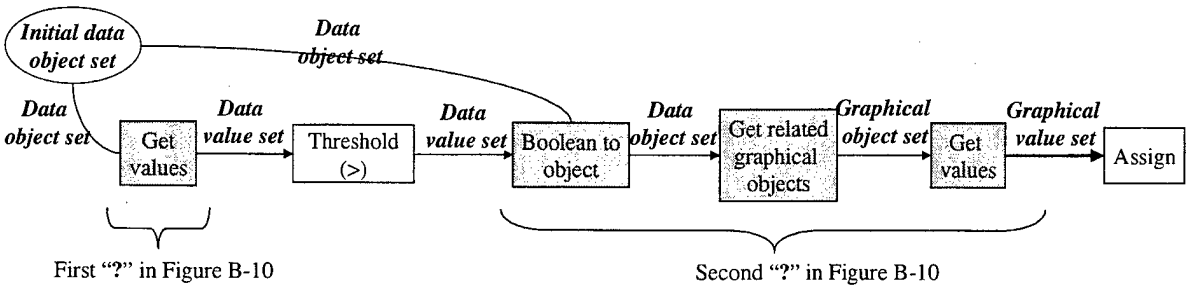


Figure B-6: Dynamic query specification with intermediate functions for inputs and outputs (Note that the visualization function classes are not shown in order to reduce the amount of diagrammatic clutter)

Figure B-7 shows the dynamic query specification with all currently unspecified function inputs in ***bold italicized*** text with light-gray background.

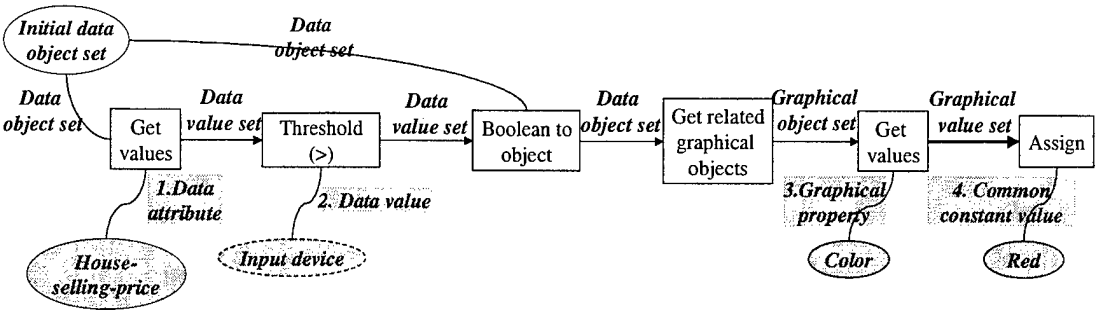


Figure B-7: Dynamic query specification with all inputs required

There are basically four necessary inputs: 1) the data attribute used to extract values for the threshold function, 2) the threshold value needed for the *threshold* object definition function, 3) the graphical property of the resulting search objects that we want to change, and 4) the graphical value we want to use as an identifying feature for all the search objects. For each of these inputs we must decide whether to provide default values (i.e. designer defined values), or whether to hook them up to an input device to get the needed values from the user. Hooking them up to a device will allow a user more flexibility in altering the functionality of the technique. On the other hand, using input devices increases the motoric load[1] of the user. In this example we have decided to provide default values for all the required input arguments except

[1] *Motoric load* (or *articulatory load*) refers to the physical effort expended by the user in manipulating physical devices such as the mouse, keyboard, or electronic pen. Example operations that result in motoric or physical effort include mouse clicks, mouse movement, key clicks, or gesturing with an electronic pen.

for the threshold value (argument 2). The default arguments are as follows: *house-selling-price* is the default search data attribute (argument 1), *color* is the feedback graphical property (argument 3), and, *red* is the common identifying color (argument 4). These arguments are shown in Figure B-7 as normal ovals.

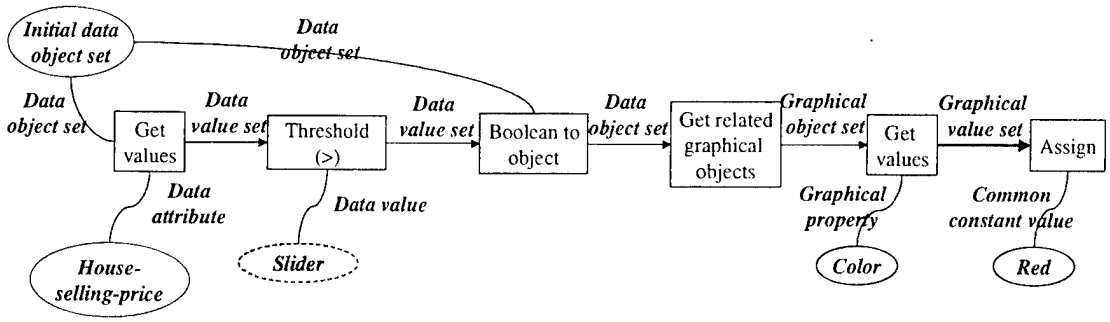
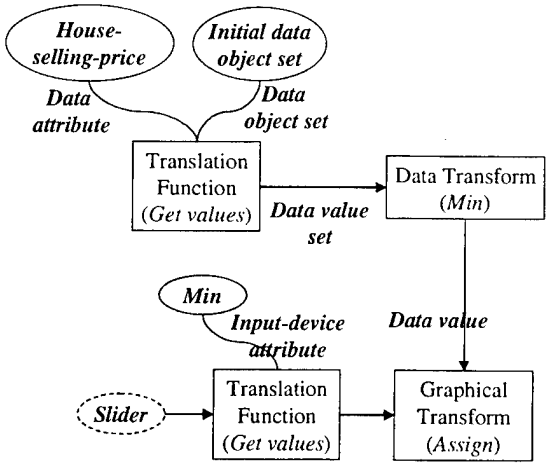
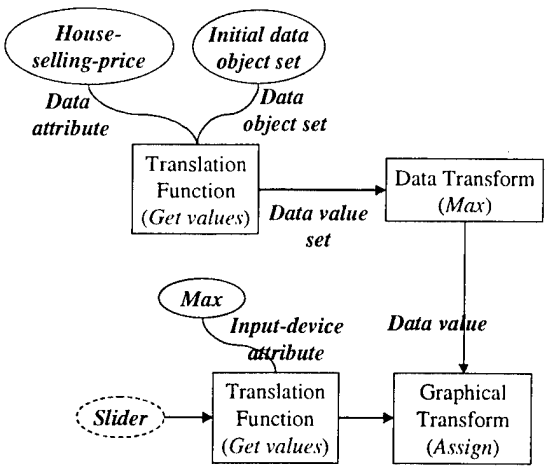


Figure B-8: Adding a slider input device for specifying the threshold constraint in the dynamic query technique

In the final steps we determine the input devices to use, and specify the initialization arguments for those devices. In this example we only have one user input value, namely the threshold value for the *greater than* object selection function. In the common dynamic query technique this input argument is attached to a *slider* input device (as is shown in Figure B-8).



(a) Specification for initializing the minimum range of the slider input device



(b) Specification for initializing the minimum range of the slider input device

Figure B-9: Initializing the *min* and *max* properties of the slider input device added in Figure B-8. The *min* and *max* values are derived by computing the *min* and *max* values of the *house-selling-price* data attribute with data transform functions.

A *slider* input device requires two initialization values, a minimum value and a maximum value, that defines the range of the slider. We set these values to be the minimum and maximum values of the *house-selling-price* attribute as is shown in Figure B-9. Initially we extract all the *house-selling-price* values from the entire data set by using the *get-values* translation function. Subsequently, we compute the *min* value using a data transformation function. This *min* value is then assigned to the *min* property of the *slider* input device which determines the minimum value on the slider range. A similar specification is used for assigning the maximum value on the slider range.

B-4 Systematic Exploration of the Instantiation Level of the Dynamic Query Slider Technique

In this section we explore the instantiation space for the dynamic query slider technique [*Ahlberg, 1994*] and discuss some of the more interesting design variations. We explore the instantiation design space by considering each of the five steps in the instantiation augmentation process (described at the beginning of this chapter III) and seeing for each step how a visualization technique can be varied:

1. Changing the specific functions used or adding more functions of the same type.
2. Changing the translation functions between object definition and transformation functions.
3. Changing how function arguments are provided (either by user or designer) as well as the default designer values.
4. Changing the type of input devices used within the design.
5. Changing how input arguments are provided to input devices.

B-4.1 Changing the Specific Functions Used or Adding More Functions of the Same Type

The first step of the instantiation augmentation process determines which specific object definition and transformation operators to use from the abstract classes described in the functional specification (e.g. functional description, graphical transform). By picking different instances of object definition and transformation functions we may generate a range of slider techniques. Figure B-10 shows the sets of alternative operators that may be used for each function class in the abstract functional specification. Different operator combinations affect the usefulness or effectiveness of the resulting technique. For example, in the slider technique shown previously, the *assign* graphical transform is used as a feedback mechanism to set the color for a group of selected or focus distributors to a perceptually salient value, e.g. *red*. Another alternative is to use the *addition* graphical transform to provide feedback by adding a constant value to the *x-position* of the focus objects thus shifting them to the right of the map. The first alternative

allows users to pre-attentively see the selected objects without losing any positional context between the focus objects and the other elements in the visualization, unlike second alternative. Nevertheless the first alternative is also more susceptible to object occlusion. The second alternative like the SDM system [*Chuah, 1995*], allows users to move a set of focus objects up to the front or up above so that they can be clearly seen without occlusion and without the noise from surrounding objects.

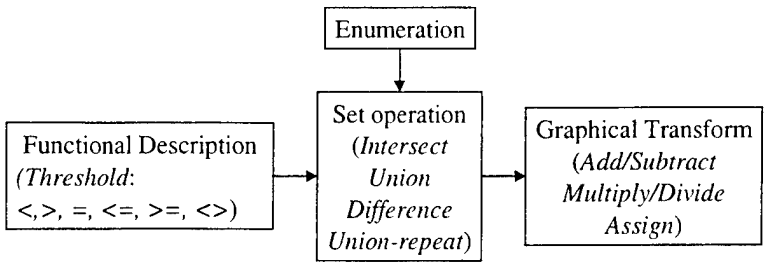


Figure B-10: Alternative object definition and transformation functions for the dynamic query slider technique

We can also experiment with expanding a function class by composing it with other functions from the same class. I.e. using multiple functional description operators or graphical transforms instead of just one. To ensure that the functional description of the technique remains unchanged however, we must only add operators that share the same general goal (e.g. *computation, summarization, feedback,* or *readability*) as the expanded operator. For example, in Figure II-13 we can use the *addition* graphical transform to alter the size of objects as well as the *assign* graphical transform to alter the color of objects. This design variation does not change the functional goal of the technique because both *assign* and *addition* functions are graphical transforms, and both functions are used for the same general goal, which is to provide *feedback* on a set of focus distributors.

B-4.2 Changing the Translation Functions between Object Definition and Transformation Functions

The second step of the instantiation augmentation phase incorporates translation functions into the design specification to ensure that the outputs of a function match the inputs of subsequent connecting functions. By using different combinations of translation functions we can vary the visualization technique design. In Figure B-11 we have enriched the dynamic query slider specification with object definition, transformation and translation functions. One variation on this design is to change the translation functions so that the *intersect* function is applied to data objects instead of graphical objects as in Figure B-12.

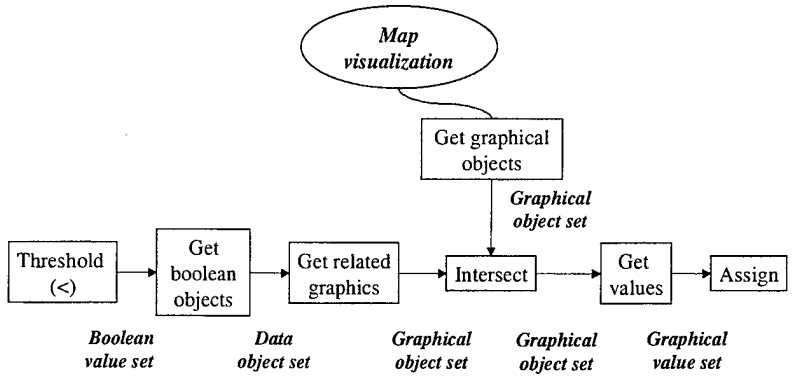


Figure B-11: Specification for the dynamic query slider technique including object definition, transformation, as well as translation functions

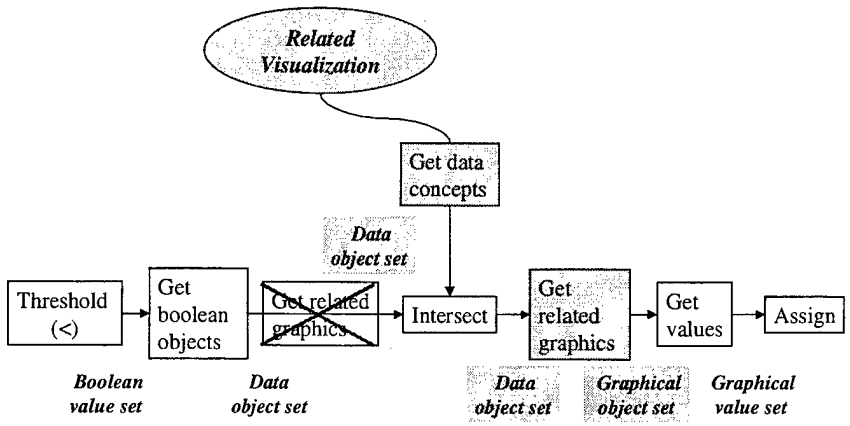


Figure B-12: In this specification we change the translation functions leading into the intersect operator so that the intersect operation is applied to data concepts instead of graphical objects as was the case in Figure B-11. Here, only those data concepts that are both selected by the *slider* and that are contained within *related_visualization* can be selected. Note that all the changes made to the specification in Figure B-11 are shown in gray in Figure B-12.

In Figure B-11 the *intersect* operator constrains the scope of the technique so that only graphical objects in the *map* region container may be colored. In Figure B-12 we constrain the scope to particular distributor data concepts instead. In this case, any graphical object may be colored as long as those objects represent data concepts that are present in *related_visualization*. In this was we can combine the object membership or query results across multiple visualization interfaces.

Translation functions may also be used to change the function arguments provided by users (through input devices) or by designers (as default values). For example, in Figure B-11 the *get-graphical-objects* translation functions is used so that the designer may enumerate the scope of the graphical object set based on a container object (i.e. the *map-visualization region*). Alternatively we may remove *get-graphical-objects* translation function and list out each individual graphical object of interest.

B-4.3 Changing How Control Arguments are Provided as well as the Default Designer Values

The third step in the instantiation augmentation process identifies the input arguments needed by the functions within a specification and connects them to a user agent or a designer agent. The specification shown in Figure B-11, for example, requires five input arguments: 1) the set of data values we want to perform the threshold operation on, 2) a threshold value for the threshold function, 3) a set of graphical objects for the *intersect* set operator, 4) a set of feedback graphical values, and 5) the feedback value used. These five input arguments are shown in Figure B-13 as text with gray background.

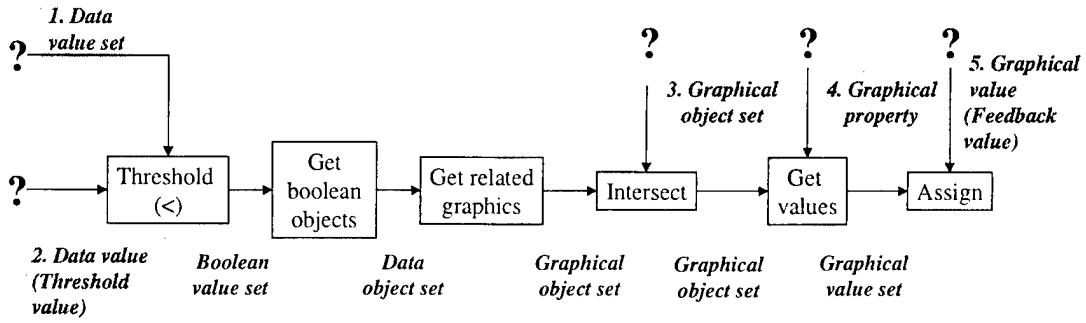


Figure B-13: This specification is similar to Figure B-11 except that here we have included all of the input arguments that are required by the various functions that have yet to be provided. Each missing argument value is indicated with a "?" symbol together with the argument type that is required. In this example there are five missing function arguments.

We can generate a variety of designs for this technique by either setting the arguments to different default values (i.e. designer defaults) or by letting users specify the arguments through input devices. One possibility is to let users provide part of the input arguments while leaving the rest as designer defaults. For example, in Figure III-12 the threshold value is provided by users through a slider input device and the set of data values for the *threshold* function are from a data attribute that is provided through a *menu* input device. All the other input arguments are provided through designer defaults. Another alternative is to give users more flexibility and let them pick the feedback graphical property (e.g. *size* or *shape* instead of *color*) and the feedback value (e.g. *blue,* or *green* instead of *red*). Note however that loading an interface with too many input controls may significantly increase the motoric and cognitive complexity placed upon users when manipulating the interface. We could also change the specification by experimenting with different default values, for example instead of using *red* as the default highlight color, we can set the highlight color to *blue* instead.

In addition to users and designers, function inputs can also be provided by other visualization techniques through composition operators as was shown in chapter II-2. However, such changes alter the

functionality of a visualization technique in significant ways, and as such their use is not encouraged during instantiation specification.

B-4.4 Changing the Type of Input Devices Used within the Design

In this step we pick the input devices for each user-provided input argument (as specified in the previous step). Suppose in the previous step we attached all the input arguments shown in Figure B-13 to input devices except input argument 3 (which feeds into the *intersect* operator). This is shown in Figure B-14 which replaces each "?" in Figure B-13 with an input device or a designer default value.

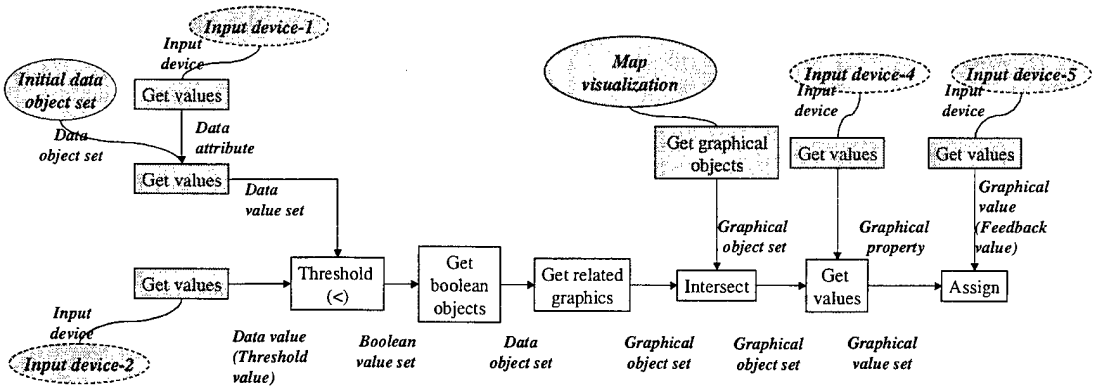


Figure B-14: Slider visualization technique with input devices

Now let us consider the types of input devices we can use to provide each of these input arguments. One way to select the threshold data attribute (i.e. *input-device-1*) is to use a *mouse* or *bounding-box* to pick an annotation object (e.g. the *x-axis*) that represents a data to graphical mapping relationship in the visualization. For example in ??, the *x-axis* annotation object represents a mapping of the *longitude* data attribute to the *x-position* graphical property. By picking this annotation object we indicate to the system that we want to perform the *threshold* operation on the *longitude* attribute. A weakness of this approach is that only graphically encoded attributes may be selected. The *mouse* and *bounding box* cannot be used to specify non-encoded data attributes because none of them are visually represented.

Other alternative input devices shown in Table III-5 include the *text-box, different menu types (option menu, scroll list, radio boxes), dial* and *slider.* These input devices are general purpose and can be used to select values, attributes, objects or containers. *Menu* input devices are especially appropriate for picking data and graphical attributes because the list of attributes is usually relatively small (< 30 attributes) and the attributes are discrete. *Dials* and *sliders*, on the other hand, are more appropriate for choosing continuous values even though they are also capable of expressing non-continuous values (e.g. alpha sliders [*Ahlberg, 1994*]). *Text boxes* are very flexible because users can type in any input argument. However, they do not

indicate which arguments are valid and which are not. More information on the expressiveness and effectiveness of input devices can be found in Card et. al.'s work [*Card, 1990*].

B-4.5 Changing How Input Arguments are Provided to Input Devices

As was discussed in chapter III-1.4, input devices have attributes as well just like data and graphical objects. Some input device attributes must be initialized before they can be used. For example the *slider* input device must first be initialized with the *min* and *max* values for the slider range. To initialize an input device attribute, we construct a visualization technique specification with the same object-definition/transformation structure as all other visualization techniques we have been discussing thus far. Therefore, we may vary an input device initialization specification by using any of the previous four steps.

In summary, we have presented five steps for systematically exploring the instantiation design space. Changes in the instantiation design space allow us to expand or change the design of existing techniques while still maintaining a common functional metaphor.

B-5 Exploring the Space of Visualization Techniques

In this section we analyze three interactive visualization techniques using the five steps in the structural augmentation process (for details refer to appendix B-4). For each technique we present its instantiation specification and describe some interesting alternative designs that can be derived from varying that specification.

B-5.1 Aggregation

The aggregation technique deals with large data sets by summarizing multiple data concepts into an aggregate concept. Aggregation may be achieved in several different ways [*Goldstein, 1994*]. In this example, we examine the aggregation technique shown in Figure B-15. In Figure B-15 users may select a set of objects using a *bounding-box*. The selected graphical objects are converted to the data concepts they represent and these data concepts are aggregated (*group objects*). In addition we also summarize a user selected attribute of the objects using the *mean* data transform function. Finally we map the new aggregate object into the visualization where the *bounding-box* was invoked (*add-object*).

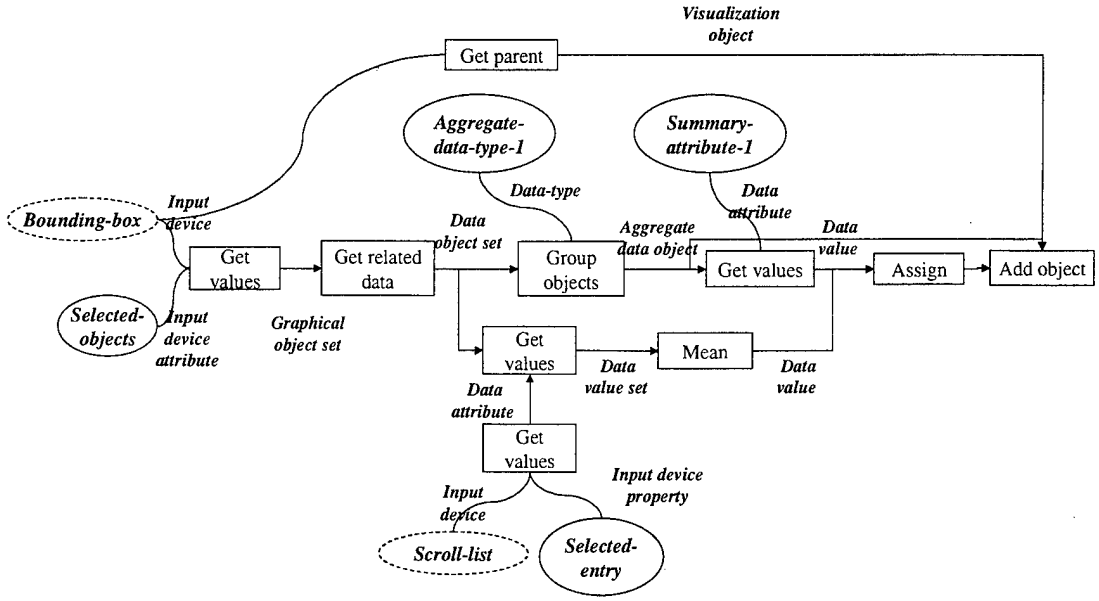


Figure B-15: One possible instantiation specification of an aggregation technique. This technique allows users to select a set of graphical objects using a *bounding-box*. It then summarizes the underlying data concepts by using a *group-objects* data transform function, thereby creating a new *aggregate-data-type-1* data-type. In addition, one of the data attributes for the selected objects is also chosen for *mean* summarization, and a new *summary-attribute-1* is created to store the mean values. This data attribute can then be mapped to a graphical property.

An interesting design variation is to use different devices to pick the aggregation objects and the visualization into which the group objects are added. This will allow us to pick objects in one visualization and have the group object appear in another visualization. Another useful variation of the aggregation technique is to remove the graphical objects that form the aggregate. Since we are adding a new group object that summarizes information on a set of individual data concepts, it may no longer be necessary to show those data concepts. Thus we add the *remove-object* mapping function that is used in tandem with the *add-object* function to reduce clutter and increase readability. Other design alternatives include adding in new input-devices for users to pick the summarization function (instead of always using *mean*), as well as the summary data-type (*aggregate-data-type-1*) and summary attribute names (*summary-attribute-1*).

B-5.2 Data Drag & Drop

Drag-and-drop is a popular input-device metaphor for moving objects from one frame or window to another. This method allows users to select an object or set of objects in an origin frame by *mouse-clicking* on them. The selected object(s) may subsequently be moved to a destination frame by *dragging* them over with the mouse. *Dragging* involves moving the mouse while having one or more of its buttons suppressed. Upon reaching the destination frame, the mouse button is released. In this work, we encapsulate the *drag-and-drop* technique as a virtual device method, just as a *bounding-box*. The *bounding-box* virtual device manages drawing of the *bounding-box*, testing and capturing all the objects encapsulated by the *bounding-box*, and erasing the box upon a mouse release event. Similarly, the *drag-and-drop* virtual device manages

storing the origin frame where the initial *mouse-click* occurred (we call this object the *click object*), the transition scenes when moving objects between frames as well as keeps track of the object where the mouse release event occurred (we call this object the *release object*).

In Figure B-16 we show the instantiation specification for a drag and drop technique that is used to transfer data concepts from one visualization to another. This *data drag & drop* technique is used extensively in the Visage visualization system [*Roth, 1996*]. Upon the *mouse release* event of the *drag-and-drop* virtual device, we query it for the *click object* and the *release object*. The *drag-and-drop* virtual device in our system returns either the graphical object that was clicked on or a container object (e.g. *region,* or *visualization*). If the mouse click occurred over a graphical object, then the graphical object is returned, if not, the device will return the smallest container upon which the click occurred. Note that in Figure B-16 we only add the *click object* into the *release visualization* if the click occurred over a graphical object and not a container object. We check for this by using the *switch* function. The *switch* function is a control operator (described in appendix A-5) that allows us to pick different streams of execution based on its inputs. Similar to the click event, the release event may occur over a graphical object or over a container object. Since we are only interested in the release visualization, we use the *get-parent* translation function to query the release object for its parent visualization container.

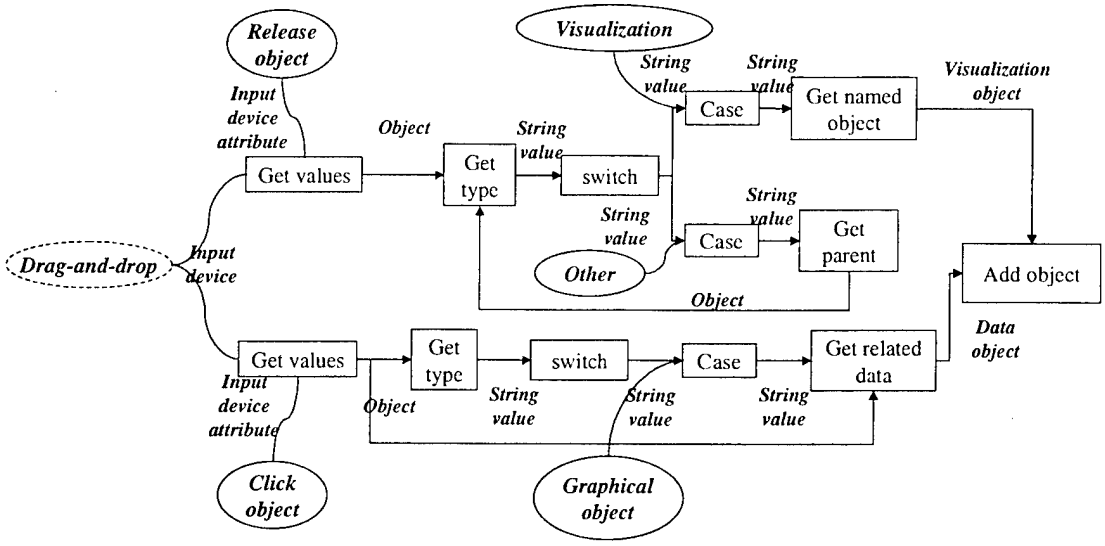


Figure B-16: Instantiation specification for Visage drag-and-drop technique. This technique allows users to pick a set of graphical objects in one visualization, and then add the underlying data concepts of the selected objects to a different visualization. The *switch* and *case* statements used above are to ensure that the user has indeed selected an origin set of graphical objects and a destination visualization container.

An interesting alternative to this instantiation design is to use different devices to pick the source graphical objects (*click object*) and the destination visualization (*release object*). For example we could use a *bounding-box* to select the initial set of graphical objects and a *mouse-click* to select the destination

visualization. This design variation illustrates that the *drag-and-drop* virtual input-device is not an integral part of the functionality achieved by the technique. Another alternative is to use pre-specified source or destination arguments. For example we could make it so that elements can be selected in any visualization but they always get added into a fixed predefined visualization. Yet another alternative is to remove the selected objects from their origin visualization (using the *remove-object* mapping operator) in addition to adding them to the *release visualization*. We could also replace the *add-object* operator in Figure B-16 with the *remove-object* operator so that instead of adding data concepts to the release visualization we are removing data concepts from it.

B-5.3 Table Lens Semantic Zoom

One very useful operation in the Table Lens system allows users to interactively control the size of table cells so that the interesting ones can be expanded and viewed in greater detail while the size of the surrounding cells are contracted so that context from these surrounding cells can still be maintained. This technique is achieved through rendering transforms that are described in detail in Rao et al.'s paper [*Rao, 1994*]. Changing the size of table cells also causes the graphical object mappings within those cells to be remapped. In particular, the larger cells will have mappings to both the *text-graphical-class* and the *bar-graphical-class* while the smaller cells will have mappings only to the *bar-graphical-class*. One way to achieve this change in mapping is through the instantiation specification in Figure B-17.

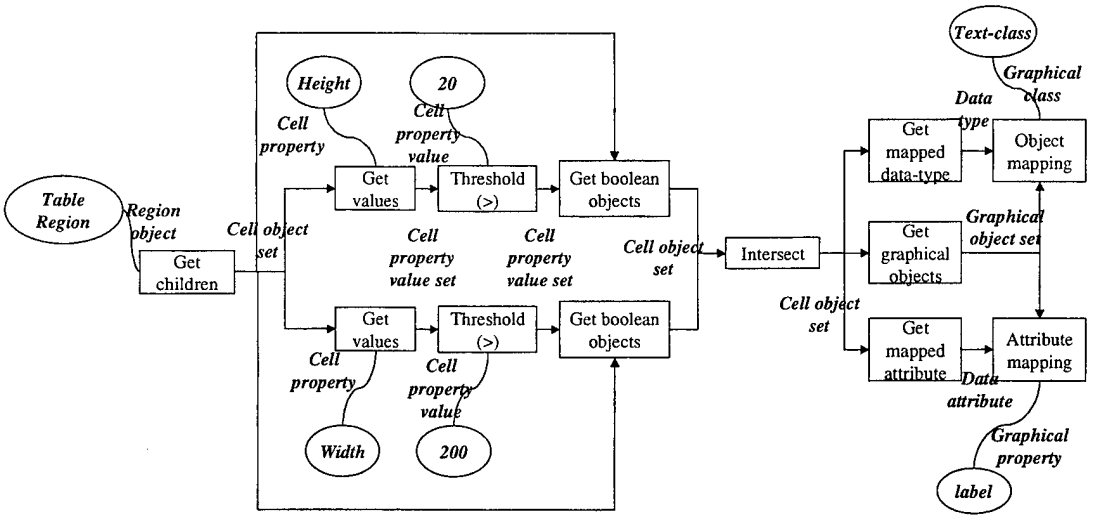


Figure B-17: Instantiation specification for the Table Lens semantic zoom operation [2]. This technique identifies the larger cells in the table (i.e. height > 20, width > 200) and adds a new *text-class* mapping for those larger cells.

[2] Note that in this specification we have introduced the *cell container object,* which we did not present in chapter III-1.1.2. This *cell container* however acts like any other container object and is placed below the *region* container in the graphical object hierarchy.

In this specification we query for all cells that have *height* greater than 20 pixels and *width* greater than 200 pixels (these are the larger cells). We then add mappings for the *text-class* and *label* properties to these larger cells. The other "small" cells are rendered according to the default mapping which in this case encodes all data concepts using the *bar-graphical-class*.

We can augment the Table Lens semantic zoom technique by adding in input-devices for controlling the *height* and *width* threshold values. This will allow users to interactively control what actually constitutes a large cell based on their perceptual abilities and the hardware screen settings. Another possibility is to add input-devices so that end-users may interactively choose the graphical property upon which to threshold on. A problem with this change is that it may de-couple the semantic zoom operation (Figure B-17) from the Table Lens size control operation because the semantic zoom will no longer be tied to the size of the cells.

Another design possibility is to break up the cells into several different groups (i.e. more than 2 as is the case in Figure B-17) and apply a different mapping to each group. For example instead of just having large and small cells, we could have large, medium-large, medium, medium-small and small groups. In this way the change in cell size will be more gradual and this may help users interpret the display changes when the *table lens* focus is changed. We could also threshold on several different graphical properties simultaneously, such as cell *size* and *color*, and have a large red colored cell group, a medium red colored cell group, a medium blue colored cell group, etc. In order to be useful however, this perceptual categorization must correspond to some meaningful data grouping.

B-5.4 Summary

These examples show some interesting design variations that can be achieved with current visualization techniques by making changes at the instantiation level. Alterations to the instantiation specification change the way with which the techniques are controlled, the amount of flexibility a user has in manipulating the techniques, as well as the amount and quality of feedback that is received. This section is provided as a contrast to chapter II-3 that explored the functional space of visualization techniques. In chapter II we explore the visualization techniques space by changing their functional semantics. Here we keep the semantics constant and explore the various structural forms that may be used to achieve the same functional capabilities and how these changes may improve the usability of the techniques.

B-6 Other Visualization Technique Issues

In chapters II and III we considered building visualization techniques either from primitives or through composition. However we only dealt with single techniques that are applied once to objects within a visualization. In practical situations, visualization techniques are often repeated many times over different

object sets. In addition, these techniques do not usually exist in isolation but are instead integrated with other visualization techniques within a common workspace. In the next two sections we begin to study these two issues and show some of the problems that may arise.

B-6.1 Repeating Visualization Techniques

Repeating interactions is a very important issue in designing interactive systems. Interactive techniques are commonly not just executed once, but are applied repeatedly to the same or to different object sets. For example throughout a data analysis session, we may want to filter and highlight different object sets within a visualization. When this occurs we must decide what happens when a previously transformed object is being transformed again. I.e. what happens when we color-highlight objects that have already been color-highlighted. There are four possible repeat alternatives: *forgetful repeat*, *additive repeat*, *incremental repeat*, and *toggle*.

A *forgetful repeat* interaction would return all previously affected values to their original state and only show the results of the new operation. For example suppose the slider technique in Figure III-11, Figure II-12, or Figure II-13 is a forgetful interaction. In this case, each time we change the slider threshold value a new set of objects will get highlighted red. All previously highlighted objects that no longer pass the new threshold will get reset back to their original color.

In contrast, an *additive repeat* interaction would add the new focus objects to the affected object set. Thus the objects highlighted red not only include the objects currently within the threshold indicated by the slider, but also those objects that have been previously selected by the slider. Newly defined objects that are already under the influence of the interaction are not changed.

The *incremental repeat* option is similar to *additive repeats* except that the interaction is applied to all input objects irrespective of whether they have already been altered. The effect of incremental repeats is the same as additive repeats for the dynamic query slider technique (e.g. Figure III-11) because the object set color is always assigned to a fixed constant value. However, incremental and additive effects are different for the technique in that colors the selected objects as well as increases their size (e.g. Figure III-13). If we used additive repeats, the selected objects are only enlarged once. However, if we used incremental repeats, the size of objects will get larger and larger as they get selected more and more times.

Finally *toggle repeat* adds selected input objects that are not already in the applied set into the applied set and removes input objects that are already in the applied set from it. For example suppose we selected a range from 100k to 150k on the dynamic query slider interface and then we selected a range from 120k to 170k. At the end of these operations, the objects that are highlighted red are those objects that are between 100k and 120k as well as 150k and 170k. The objects between 120k to 150k get un-highlighted because they fall within both of the chosen ranges.

This issue of repeating operations has been dealt with by Wills [*Wills, 1996*], but he focused only on object selection techniques. In order to deal with a wider range of techniques we added the *incremental repeat* option that is missing from Wills' framework. Wills did not consider this option because applying multiple object selections with the *additive repeat* option or the *incremental repeat* option has the same effect. This is because the selection technique only provides users with binary feedback (either an object is colored to indicate that it has been selected or not colored to indicate that it is not selected). For techniques that provide non-binary feedback (e.g. size increase, position shifts, etc), the *additive repeat* and the *incremental repeat* options will have different effects as was discussed above.

B-6.2 Integrating Visualization Techniques within a Common Workspace

When we integrate several techniques within the same environment, we must ensure that they are consistent with each other. A consistent workspace such as the Mac or the Windows environment allows users to access the techniques within it with greater ease because once users understand the metaphor or "physics" of that workspace, they can easily pick up on how new techniques would work within that space. Trying to develop a consistent set of rules for visualization technique design, however, is often a difficult and protracted process. Some systems base their workspace upon physical metaphors from the real world. The advantage of this approach is that most people understand the basic physical laws of the real world very well and thus are able to apply that knowledge to the virtual workspace. The disadvantage is that if we follow the "real world" too closely, we become constrained by its limitations and thus may not use the flexibility afforded by the computer media to its fullest. That is why most visualization systems turn out to be hybrid systems consisting of both "real world" and "virtual world" rules. Examining different visualization technique and workspace metaphors and deciding on the right balance of "real world" and "virtual world" rules is a very interesting but large area of study. Such considerations are beyond the scope of this thesis and is left for future work.

Another very important issue to consider when integrating a set of techniques into an environment is whether they conflict with one another. As a first step towards dealing with visualization technique conflicts, we consider all possible inconsistencies that may occur between any pair of primitive visualization techniques by sensing for conflicts based on five important dimensions: the object definition set, the transformation function used, the graphical property or data attribute used with the transform function (if applicable), the graphical or data value(s) used with the transform function (if applicable), and the input devices attached to the technique (if applicable). Based on these dimensions we identified the following conflicts:

- *Resource Ambiguity*: This conflict arises for different techniques[3] that use the same feedback value and properties. For example, consider a system that allowed users to highlight objects red either by

[3] Two techniques are considered different if they have different instantiation specifications.

painting with a bounding box or querying with a slider. In such a system there is ambiguity as to how/why a red object got highlighted. Problems arise because when the primitives are different, users commonly also expect different feedback or results from them. By using the same feedback property and value, users can easily get confused as to which technique caused a particular change in the display.

- *Resource Overload*: This conflict arises when two different techniques change the same data attribute or graphical property in different or opposing ways. For example, consider a system that used sliders to highlight objects red and bounding boxes to highlight objects blue as in the painting systems. When an object falls within the applied set of both techniques, there is uncertainty as to which color should be used.
- *Resource Adjacency*: This conflict arises when two spatially adjacent visual components are changed in the same way. For example, consider two spatially adjacent components within a mark graphical object: 1) the mark outline and 2) the mark body. When both components are colored in the same way, we lose our ability to perceive the boundary of separation between them. This raises complications when the outline width is also used to represent a data attribute because we lose this data encoding whenever the outline and body color of a mark object coincide.
- *Resource Conjoins*: This conflict arises when two resources are conjoint parameters. Conjoint parameters combine together to produce emergent properties. Examples conjoint properties include the width and height of a rectangle that produces the "area" emergent property. When one conjoint parameter is changed without comparative changes in the other associated conjoint parameter(s), the emergent property shown may no longer be correct and this may cause users to misinterpret the data.
- *Resource Inconsistency*: Resource inconsistency refers to all syntactic errors in a visualization technique specification. Some example syntactic errors include using incorrect types of input arguments, or trying to use an invalidated resource. For example, suppose a *remove-objects* visualization function is used to delete certain graphical objects from a visualization. A resource inconsistency error arises if a later visualization function tries to change those deleted objects.

Dealing with conflicts in a set of visualization techniques is a difficult process because it depends on the visualization, the techniques used, and the task. If the task does not require persistence in the visualization technique effects, then conflicts with that technique may not be as important. Otherwise, we must find some way to resolve the collisions and this may get very complex when there are many techniques within the environment. We therefore leave the problem of conflict resolution for future work.

Appendix C

Appendix to Design Heuristics (Chapter IV)

C-1 GOMS Evaluation for Airline-Scheduling Task in Chapter IV-1

Given an origin and a destination city, the user "attempts to locate the two flights arriving in and departing from a layover city that offer the minimum amount of `down time' between the flight times and the beginning and ending time of a scheduled meeting (in the layover city)".

In the following example designs, suppose that the origin and destination cities are *Los Angeles (LAX)* and *Boston (BOS)* and that the layover city is *Chicago (ORD)*. Further, suppose that the meeting time is from 2 p.m. to 4 p.m. Our airline scheduling data set has a total of 135 flights. 47 flights originate from Los Angeles (LAX) and 18 of those go to Boston (BOS). Of those 18 flights, only 10 fulfill the meeting time constraints.

C-1.1 GOMS Evaluation for the Cognitive Solution of the Airline Scheduling Task

Flight number	Origin city	Destination city	Flight arrival time	Flight departure time
United_652	ORD	EWR	5:32 a.m.	2:30 a.m.
NW_826	MEM	ATL	10:13 a.m.	8:00 a.m.
Trans_235	MDW	LGA	9:00 a.m.	5:55 a.m.
United_244	LAX	ORD	3:37 p.m.	9:45 a.m.
United_342	ORD	MCO	12:19 a.m.	8:55 a.m.
American_494	ORD	EWR	7:48 a.m.	4:30 a.m.
American_491	BOS	DFW	7:29 a.m.	4:15 a.m.
• • •				
United_7282	JFK	BOS	8:31 a.m.	7:00 a.m.
NW_5875	MEM	MOB	9:45 a.m.	8:20 a.m.
Delta_725	DFW	SFO	10:54 a.m.	9:00 a.m.
TWA_7762	PHL	JFK	7:53 a.m.	7:04 a.m.
American_346	ORD	LGA	10:04 a.m.	7:00 a.m.
American_340	ORD	LGA	11:36 a.m.	8:30 a.m.
Continental_1943	AH	LAS	9:46 a.m.	8:39 a.m.

Figure C-1: Cognitive design for the airline-scheduling task
(Note that the flights are not all shown here because the table is very large)

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	**Sub total (msec)**	**Total time (msec)**	Target objects (data or graphics)
Find flight originating from Los Angeles (LAX) and point left hand at flight	Attend find flight origin	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read origin city name	290			
	Initiate comparison	50			
	Compare origin city name with LAX	50			
	Verify comparison	50			
	Sub-total		570		
Find destination of flight and determine whether it is Chicago (ORD)	Attend find flight destination	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read destination city name	290			
	Initiate comparison	50			
	Compare destination city name with ORD	50			
	Verify comparison	50			
	Sub-total		570		
Get flight arrival time	Attend get flight arrival time	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read arrival time	290			
	Attend compare time with start of meeting	50			
	Compare if time is before meeting start time	50			
	Verify compare	50			
	Sub-total		570		
Compute time before meeting	Attend compute time before meeting	50			
	Subtract flight arrival time from 4 p.m. (3 significant figures) 150 + (*n*-1) * 100	350			
	Verify time before meeting	50			
	Sub-total		450		
Determine if current prior meeting downtime is the minimum	Attend compare with current min downtime	50			
	Compare with current min downtime (Assume an average to 2 significant figure comparison = 50 msec * 2 = 100 msec)	100			
	Verify results	50			
	Sub-total		200		

Time taken to move finger in the calculation below we estimate as follows:
We assume that the visualization is enlarged so that each label entry is at least the width of a finger. Or else, finger pointing would be very difficult. Thus,
Height of widest finger = Height of each entry = 0.5 inches
For this particular design and data set, we assume that 5 out of the 10 flights that fulfill all time and city constraints require min changes. If they are evenly spread out, the movement per min number is 135/5 steps = 27 * 0.5 inches = 13.5 inches
Thus using Fitts Law, the estimated time for movement is 100 * ($\log_2$ (13.5/0.5)+.5) = 525 msec

Note that for simplicity we assume that all the flights fit within 1 screen. This is highly unlikely since there are 135 flights. However even with this simplifying assumption the estimated time for this cognitive solution is still very significant.

If so, point to current row with right hand and keep current min downtime in STM	Attend change finger positions	50			
	Initiate finger lift	50			
	Lift finger	60			
	Initiate finger move	50			
	Move finger 100 * ($\log_2$ (13.5/0.5)+.5)	525			
	Initiate finger drop	50			
	Drop finger	60			
	Sub-total		845		

Total time for processing all rows for flight before meeting include:
Time taken to process origin of all rows = 135 * 570 msec. = 76950 msec.
Time taken to process destination of all rows. This only applies to flights with origin from LAX of which there are 18 = 18 * 570 msec. = 10260 msec.
Time taken to process total downtime before meeting. This only applies to flights from LAX to ORD of which there are 18 = 18 * 570 msec. = 10260 msec.
Time taken to process min-downtime. This only applies to flights from LAX to ORD that arrive before the meeting of which there are 10 = 10 * (450 + 200) msec = 6500 msec.
Time taken for finger movement. This depends on the number of times we have to change the min entry. Since there are 10 flights which fulfills both city and time constraints, we assume half of these require min changes thus total time = 5 * 845 msec. = 4225 msec.

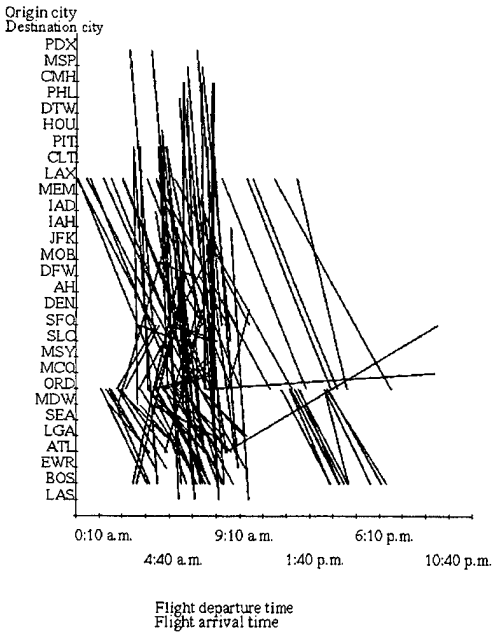
Total time taken for processing flight with minimum total before meeting downtime	76950 + 10260 + 10260 + 6500 + 4225			**108195**	

Repeat for getting total downtime after meeting. Total time for processing all rows for flight after meeting include:
Time taken to process origin of all rows = 135 * 570 msec. = 76950 msec.
Time taken to process destination of all rows. This only applies to flights with origin from ORD of which there are 47 = 47 * 570 msec. = 26790 msec.
Time taken to process total downtime before meeting. This only applies to flights from ORD to BOS of which there are 19 = 19 * 570 msec. = 10830 msec.
Time taken to process min-downtime. This only applies to flights from ORD to BOS that depart after the meeting of which there are 10 = 10 * (450 + 200) msec = 6500 msec.
Time taken for finger movement. This depends on the number of times we have to change the min entry. Since there are 10 flights which fulfills both city and time constraints, we assume half of these require min changes thus total time = 5 * 845 msec. = 4225 msec.

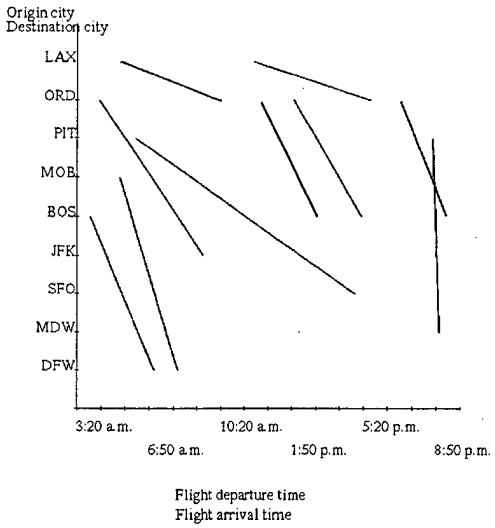
Total time taken for processing flight with minimum total after meeting downtime	76950 + 26790 + 10830 + 6500 + 4225			**125295**	
Add both downtimes (We assume that both downtimes can be stored in STM)	Attend add downtimes	50			
	Mental add (Assume 3 figures) (150 + (n-1) * 100)	350			
	Verify results	50			
	Sub-total	450			
Total time	108195 + 125295 + 450			**233940**	

Total time taken to solve the airline-scheduling task using Figure C-1 is 234 seconds or approximately 4 minutes.

C-1.2 GOMS Evaluation for the Pure Perceptual Solution of the Airline Scheduling Task



(a) Full data set
This visualization shows all the elements in the data set (i.e. all 135 flights).



(b) Truncated data set.
This example visualization shows the ideal case where there is little occlusion among the different flight lines. This data set was chosen so that it contains some flights that fulfill the task constraints as well as some other random flights that do not occlude one another.

Figure C-2: Perceptual design for the airline-scheduling task
Each line represents a flight with origin and destination city mapped onto the *y-axis* and arrival and departure time mapped onto the *x-axis*. This is the best design that gets generated when ONLY mapping operations are considered by the automatic system. I.e. this is the best possible design from current state of the art systems.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	**Sub total (msec)**	**Total time** (msec)	Target objects (data or graphics)
Scan on *y-axis* to find origin city (LAX). We assume that cities are ordered alphabetically, thus a binary type search can be applied. Since there are 29 cities, a total of $\log_2(29) = 5$ searches are necessary to get to the desired city	Attend find origin city on *y-axis*	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read city name	290			
	Initiate comparison	50			
	Compare origin city name with LAX	50			
	Verify comparison	50			
	Sub-total		570		
	Total time (5 * 570)		2850		
Scan to the right from *origin* city position and process next line that start from this *y-position*. Specifically scan to end point of line and determine if it is from ORD as well as before the meeting start time (we assume that the start and end meeting times as well as the origin, layover, and destination cities are marked on the display or stored in STM by the user).	Attend process line end-point	50			
	Initiate eye movement	50			
	Eye movement to end of line	30			
	Perceive point	100			
	Is point arriving at destination city?	50			
	Verify point arriving at destination city	50			
	Sub-total		330		
	Is point before meeting start time?	50			
	Verify that point is before meeting start time	50			
	Sub-total		100		
Compare end-point with current best candidate flight (here we determine if the current point is before or after our finger position which is at the current best flight candidate).	Attend compare *x-distance*	50			
	Is point after finger position?	50			
	Verify results	50			
	Sub-total		150		
The distance moved here depends on the x-distance between different flights before the meeting. This distance is NOT dependent on number of elements as was in the previous example but is instead dependent upon the x-axis scale. Assuming that the visualization fills the entire screen, the maximum x-axis length would be 14 inches in a 21 inch CRT display screen. Since 5 moves are necessary, we assume an average of 2.8 inches per move					
If so, point to current row with left hand	Attend change finger positions	50			
	Initiate finger lift	50			
	Lift finger	60			
	Initiate finger move	50			
	Move finger $100 * (\log_2 (2.8/0.5)+.5) =$ $299 =$ approx 300	300			
	Initiate finger drop	50			
	Drop finger	60			
	Sub-total		620		

Total time for processing all rows include:
Time taken to get to the proper origin city position on the *y-axis* = 2850 msec.
Time taken to process destination of all lines that start from the origin city (LAX) of which there are 18 = 18 * 330 msec. = 5940 msec.
Time taken to process total downtime before meeting. This only applies to flights from LAX to ORD of which there are 18 = 18 * 100 msec. = 1800 msec.
Time taken to determine if flight is best candidate. This only applies to flights from LAX to ORD that depart after the meeting of which there are 10 = 10 * 100 msec = 1000 msec.
Time taken for finger movement. This depends on the number of times we have to change the min entry. Since there are 10 flights which fulfills both city and time constraints, we assume half of these require min changes thus total time = 5 * 620 msec. = 3100 msec.

Total time taken for processing flight with minimum total before meeting downtime	2850 + 5940 + 1800 + 1000 + 3100			**14690**	
Repeat for getting total downtime after meeting. Total time for processing all rows for flight after meeting include: Time taken to get to the proper origin city position on the *y-axis* = 2850 msec. Time taken to process destination of all lines that start from the layover city (ORD) of which there are 47 = 47 * 330 msec. = 15510 msec. Time taken to process total downtime before meeting. This only applies to flights from LAX to ORD of which there are 19 = 19 * 100 msec. = 1900 msec. Time taken to determine if flight is best candidate. This only applies to flights from LAX to ORD that depart after the meeting of which there are 10 = 10 * 100 msec = 1000 msec. Time taken for finger movement. This depends on the number of times we have to change the min entry. Since there are 10 flights which fulfills both city and time constraints, we assume half of these require min changes thus total time = 5 * 620 msec. = 3100 msec.					
Total time taken for processing flight with minimum total after meeting downtime	2850 + 15510 + 1900 + 1000 + 3100			**24360**	
Get exact downtime before meeting	Attend get exact downtime				
	Attend lookup arrival time from *x-axis*	1135			
	Mental subtract arrival time from meeting start time (150 + (n-1) * 100)	350			
	Verify downtime	50			
	Sub-total		1535		
Get exact downtime after meeting		1535			
Add both downtimes	Attend add	50			
	Mental add	350			
	Verify results	50			
	Sub-total	450			
Total time	14690 + 24360 + 1535 + 1535 + 450			**42570**	

Total time taken to solve the airline-scheduling task using Figure C-2 is approximately 43 seconds.

C-1.3 GOMS Evaluation for the Perceptual + Data Computation Solution of the Airline Scheduling Task

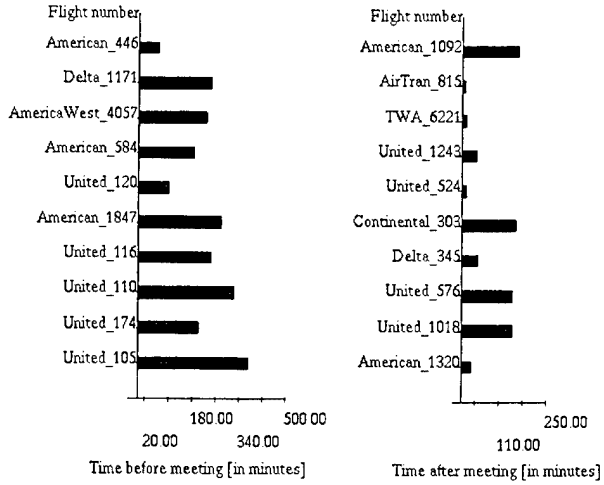


Figure C-3: Design generated when data processing operations are integrated into the automatic visualization system. The full data set is considered here but data transforms are applied by the automatic system to filter the data set so that only relevant flights are shown. The total downtime before the meeting for the flights from LAX to ORD is shown on the left chart and the total downtime after the meeting for the flights from ORD to BOS is shown on the right chart.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	**Sub total (msec)**	**Total time** (msec)	Target objects (data or graphics)
Find shortest bar in first chart	Attend find shortest bar	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive shortest bar	100			
	Verify shortest bar	50			
	Sub-total		280		
Lookup total downtime for that bar	This figure was computed from Design 4, Task 2 in appendix E-2.4	1135			
Find shortest bar in second chart		280			
Lookup total downtime for that bar		1135			
Add both downtimes (assume 3 significant figures)	150 + (n – 1) * 100	350			
Total time	280 + 1135 + 280 + 1135 + 350			**3180**	

Total time taken to solve the airline-scheduling task using Figure C-3 is approximately 3 seconds.

C-2 Airline-Scheduling Task Design Alternatives

In chapter IV-1 we discussed several design alternatives for the airline-scheduling task that consists of different blends of data and mapping transforms. In this section we present several more design alternatives

and discuss their strengths and weaknesses to highlight some of the issues that arise when we are trying to make decisions about whether to use data or mapping transforms to solve user goals. In particular we want to illustrate that it is not always best to fully pre-compute all tasks and the best design often is one which consists of a combination of both data computation and perceptual mapping operations.

In chapter IV-1 we focussed on design examples that assumed complete task specification (i.e. all the task parameters, e.g. *origin* and *destination cities*, *start* and *end meeting times*, are known before the analysis). In these cases, the data computation solution performs very well. However when tasks cannot be fully captured at the outset, it becomes necessary to map more of the data to graphics (i.e. it becomes more difficult to use data computation solutions) as was shown in Figure IV-5.Another way to deal with low task specificity (particularly imprecise task parameters) is to add input devices into the design and let users specify the task requirements during the data analysis process. In Figure C-4, we use the same data computation design as in Figure IV-3, however here, we assume that specific knowledge of the task parameters (e.g. the *layover city* and *meeting time* information) is unavailable. To enable data computation we integrate the graphic design with an option button for specifying the *layover city* and two sliders for specifying the start and end meeting times.

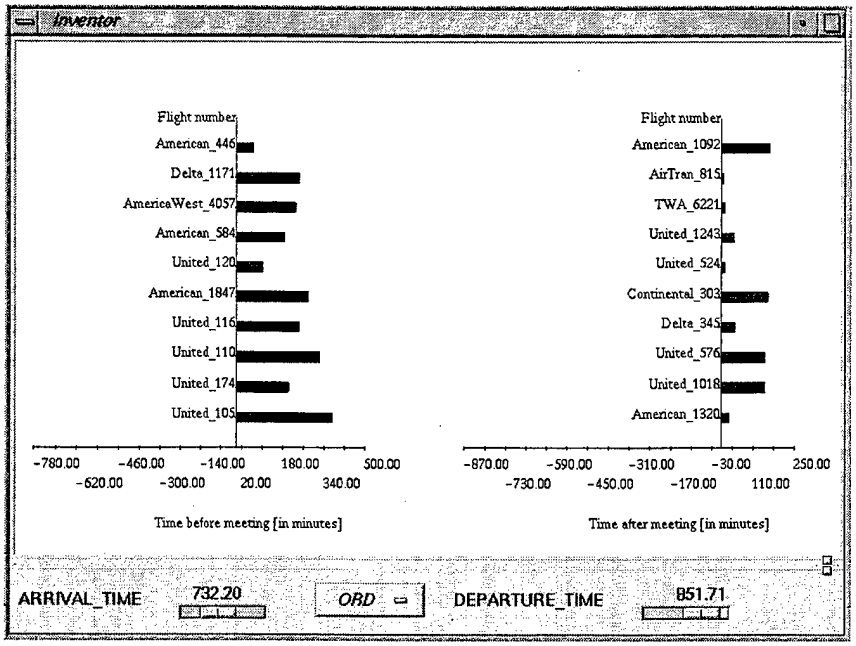


Figure C-4: Solving the airline schedule task with input devices. Here we assume that some of the airline-scheduling task constraints are unknown (i.e. the *layover-city* is not known and the *beginning* and *ending* meeting times are also not known). These constraints can be entered into the system through *sliders* and *option-buttons*.

Through these input devices, users may test different sets of input task parameters. Each different test, however, requires manipulation of some or all of the devices. In addition, only results from a single test are shown at any one time, thus it is highly probable that users may forget results from previous tests and need

to repeat a test several times. These factors increase the difficulty and time required for solving the task. Therefore if the user is really unsure of the task parameters, or if the user foresees that the particular task step may be repeated many times, it is more effective to encode the information by mapping all the data to graphics so that the task can be performed perceptually rather than through data computations and summarizations coupled with input devices. For example Figure IV-1 encodes all of the airline scheduling data with mapping techniques, thus it does not require the system to know any of the task parameters (i.e. no user input is required even if we are unsure of the task parameters). Figure C-5 filters out all the flights based on *origin, destination,* and *layover* cities, but show the *arrival* and *departure* times of the relevant flights, so that there is less clutter compared to Figure IV-1, but some flexibility in our task meeting time constraints.

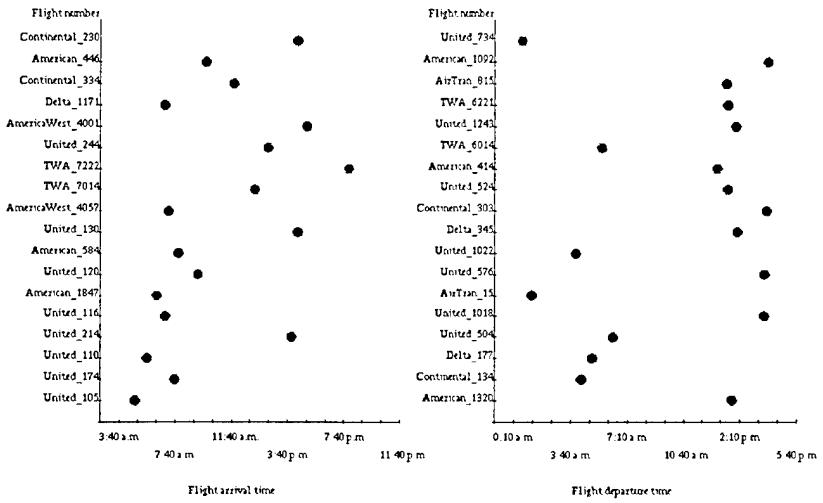


Figure C-5: Design alternative for airline-scheduling task where flights are filtered based on origin, destination, and layover city information. Arrival and departure times are shown however to allow flexibility in our task meeting time constraint. *Flight_arrival_time* is mapped to the *x-axis* of the left chart and *flight_departure_time* is mapped to the *x-axis* of the right chart.

Another design alternative for our airline-scheduling task is to combine the results of both the flights to and from the *layover city* and compute the total downtime for all flight pairs as in Figure C-6. The advantages of this display are that users need not compute the *total downtime* perceptually and that the *total downtime* can be determined with better accuracy compared to Figure IV-3. This is because in Figure C-6 the user only needs to look up the bar ends, whereas in Figure IV-3 the user is required to estimate the sum lengths of two spatially separated bars.

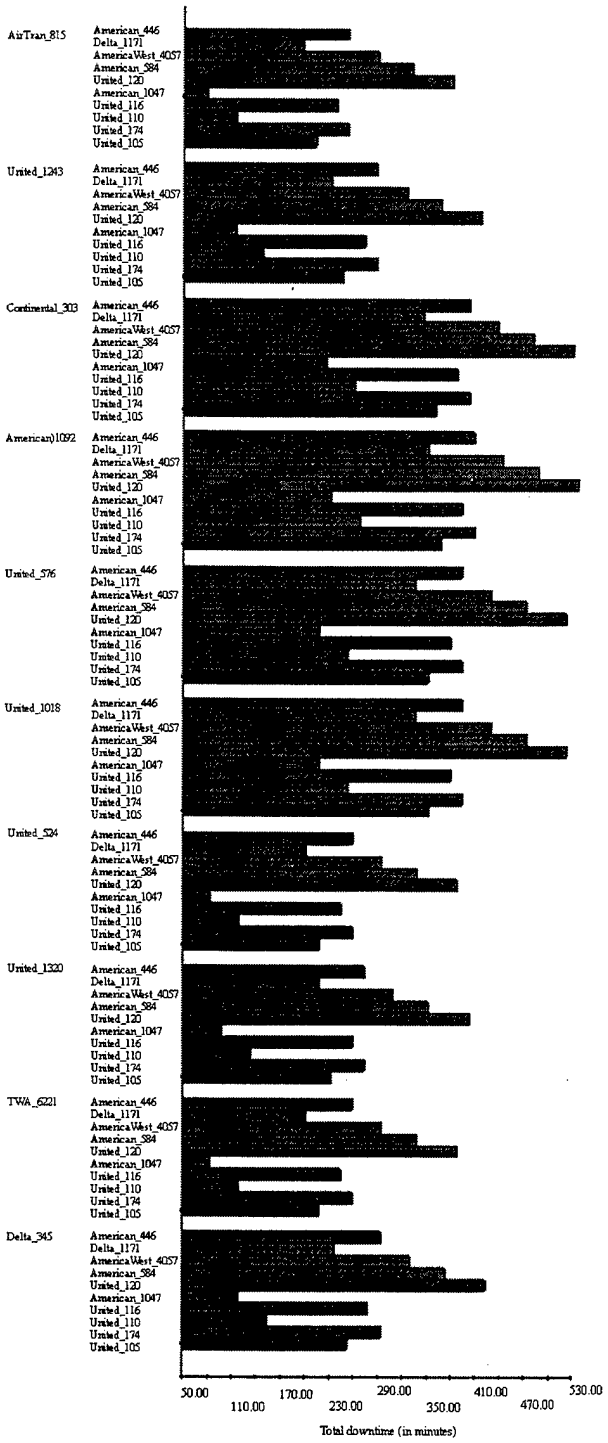


Figure C-6: Solving the airline-scheduling task where all searches as well as the total-downtime computations are performed with data transform techniques. *Total_downtime* is pre-computed and mapped to the *x-axis*. Note that in this design there are many more graphical elements than the other airline-scheduling task designs because all possible pairs of flights must be considered and shown.
[Note: the indented labels for this design was manually generated]

A weakness of Figure C-6 however, is that all flight pairs must be listed. Consequently if there are *n* flights before the meeting and *m* flights after the meeting we would need to show *n* * *m* data points, whereas in Figure IV-3 we would only need to show *n* + *m* data points. These extra data points make Figure C-6 much larger than Figure IV-3 and thus it requires more screen space (Figure IV-3 only shows 20 bars while Figure C-6 shows 100 bars). For larger data sets, the visualization in Figure C-6 might not even fit within a CRT screen. In such cases, we can only view a subsection of the visualization at a time and must navigate, using input devices, to different sections of the visualization. The manipulation cost involved in navigating and the cognitive cost involved in managing the different information slices make finding the two shortest bars in Figure IV-3 more efficient than Figure C-6 for any reasonably sized data set (i.e. > 30 elements).

C-3 Task Model

In this thesis, we focus on tasks in exploratory data analysis (EDA). EDA was first introduced by Tukey in his book entitled *Exploratory Data Analysis* [*Tukey, 1977*], where he described a series of logical and paper/pencil techniques for processing data. Since then several task models have been developed that try to characterize and define the operations involved in EDA. Some of these task models break high-level data analysis operations (e.g. cross-examine, discover shortage) down to simpler logical operators (e.g. find, compute, look-up). A designer or an automatic visualization system can then offload some of these logical operators onto the human perceptual system by generating a suitable external representation. Another alternative is to off-load some of the logical operations onto the computer system by using data computation functions (Figure C-7).

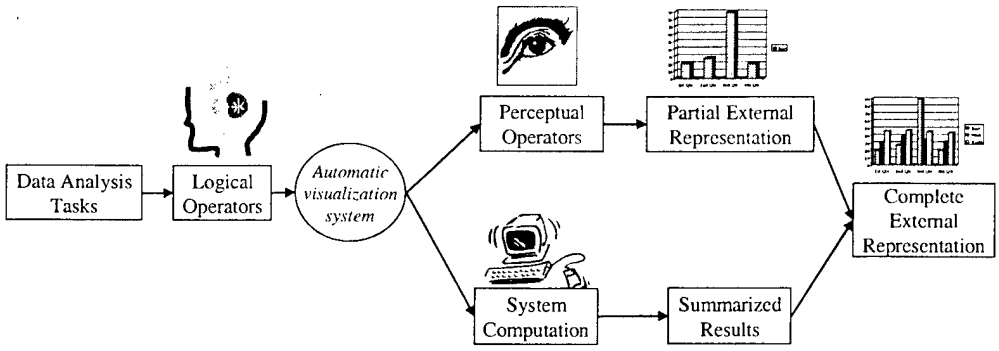


Figure C-7: This diagram shows how data analysis tasks can be broken down into perceptual operators (mapping transforms) and system computation operators (data transforms) and how these operators ultimately combine to produce a visualization design (external representation).

Our task model most closely resembles the task models presented by Casner and Senay et. Al [*Casner, 1991; Senay, 1994*]. Casner presented two logical task classes: *search* and *computation*. Search operators include *search*, *lookup*, *search and lookup*, as well as *lookup and verify*. Computation operators include *equal*, *less than*, *greater than*, *addition*, *difference*, *multiplication* and *quotient*. Senay and Ignatius

presented three basic logical operations: *search*, *look-up*, and *compare* that are very similar to Casner's logical tasks. However, they extended the task specification to deal with groups of objects.

Our task model has four logical task operators similar to the ones in Casner and Senay et al.'s task language, *look-up, compute, find* and *compare*. We also expand our task language from previous approaches in three primary areas: 1) allowing task embeddings, 2) introducing precision declarations within the task, and 3) providing mechanisms for capturing task iterations.

C-3.1 Logical Tasks (Logical Operators)

Table C-1 summarizes our four simple logical operators: *lookup*, *compute*, *compare*, and *find,* and shows their input and output arguments. These tasks can be applied to single data objects and values or to sets of data objects and values. Tasks may also be applied to unknown arguments (indicated by a '?') if the user is unsure of an input argument during specification.

	Input arguments	Output arguments
Lookup	Data object(s), Data Attribute	Data value(s) or Data object(s)
Compute	Compute operator (addition, difference, etc), Data value(s), Data value(s).	Data value(s)
Find	Relationship, Data value(s), Data value(s).	Data object(s)
Compare	Compare operator (>, <, =, >=, <=), Data value(s), Data value(s).	Relationship

Table C-1: Tasks and their input and output arguments

By basing our task model on previous work in exploratory data analysis, we ensure that our operators are expressive of a relatively wide range of EDA goals. In the next sections we describe the four logical operators in Table C-1 in greater detail.

C-3.1.1 Lookup

The lookup task gets attribute information on one or more data concepts. It has two input arguments: the data concepts to perform the lookup on, and the data attribute of interest. For example, getting the *selling price* of a house is a lookup task. In this scenario, the lookup object is *house-1* and the lookup attribute is *selling-price*. The result of this lookup task is a data value indicating the selling price of *house-1*.

```
(Lookup      house-1, selling-price )
```

We can also look up attribute values for a set of data concepts. For example, in the task specification given below we have applied the lookup task to a set of three houses. In this case, the lookup task will return a set of three data values corresponding to the selling prices of the three input house concepts.

```
(Lookup      {house-1, house-2, house-3}, selling-price )
```

In addition to data values, we may also look up data relationships. For example, we may look up the *owner* relationship for a set of houses. This relationship points to a set of *person* concepts that represent the people who own the input houses. In this case the lookup task returns a set of data concepts rather than a set of data values as was in the previous two cases.

```
(Lookup      { house-1, house-2, house-3 }, owner  )
```

The lookup task is the most basic of all the logical operators. It is often embedded within one of the other logical tasks because they operate on data values, which must first be extracted or looked-up from data concepts.

C-3.1.2 Compute

The compute task generates new data values based on existing information. For example, the *difference* compute operator can be used to derive *gross-profit* values from *total-sales* and *total-cost*, as is shown below.

```
(Compute     Difference, (Lookup company-1, total-sales),
                         (Lookup company-1, total-cost) )
```

A compute task was also used in the airline-scheduling example presented in chapter IV-1 to calculate the *total downtime* in the *layover city*. In this case, the *addition* operator was applied to the *time before meeting* and the *time after meeting* data values.

We can also use computes to summarize a set of data values by determining their *minimum, maximum, mean,* or *median*. In this case the input arguments include the summarize operator and the set of values to be summarized. The output will be one data value (irrespective of the input data set size). For example we may use the specification below to determine the average *selling price* for a set of houses.

```
(Compute     Mean,
             (Lookup  { house-1, house-2, house-3 },
                      selling-price ))
```

The type of computation operators we consider in our work include all the *data transform techniques* presented in chapter III.

C-3.1.3 Find

Find refers to the task of looking for a set of data concepts that fulfills certain data constraints. Some common constraints include upper and lower bound constraints (<, >), equality constraints (=) or both (<=,

>=). For example we may want to find all houses whose *selling price* exceeds 100k (lower bound constraint).

```
(Find    >,
         (Lookup { house-1, house-2, house-3 } , selling-price),
         100k)
```

Note that find is a logical task and it does not necessarily mean visual search. The find task *can* be solved through visual search by mapping the find attribute, (e.g. *selling price*) to a visual property (e.g. *x-position*). The find task can also be accomplished through data transform techniques (without the need for any visual search), in which case it would take the form of a data query. In the airline-scheduling example described earlier, there were several find tasks within the task sequence such as "finding all the flights originating from *Los Angeles*". To fulfill this task we can visually search for all such flights using Figure IV-1. Alternatively in Figure IV-3, the find task is performed by the system and the user is only presented with the flights that fulfill all of the find constraints.

Another common find task is to compare two sets of data values. For example, we may want to find all the months in which the price of rice exceeded the price of wheat. In this case we are comparing pairs of values for each month concept and returning all the concepts (i.e. months) whose value pair fulfills our constraint.

```
(Find    >,
         (Lookup     { month-1, month-2, …, month-n } ,
                     rice-selling-price ),
         (Lookup     { month-1, month-2, …, month-n } ,
                     wheat-selling-price ) )
```

The find tasks that we have discussed so far are simple find tasks involving the comparison of value pairs. More advanced find tasks may look for complex group relationships among a set of values. For example, we may want to find all the months where the price of rice is increasing. In this case, we are looking for a set of values, related so that each value is greater than the previous one. The increasing trend is just one of many possible trends such as cycles, bell-curves, etc. Finding data trends is often difficult to achieve with data computation functions because the rules for capturing trend behavior are complex (i.e. low task specificity). For example, consider Figure C-8, which shows a linearly increasing trend with some outliers. A strict greater-than rule would not work here because the outlier points do not fulfill that rule. To capture this trend, we need a more complex rule allowing for outlier points and other similar exceptions. In addition, just specifying the trend as increasing may be insufficient. We may need to specify other trend properties such as its rate of increase, allowable error rate, number of outliers permitted, etc. Thus, there is a great deal involved in conveying to the system the exact user requirements for finding trends, thereby making them more appropriate for perceptual analysis (i.e. mapping techniques).

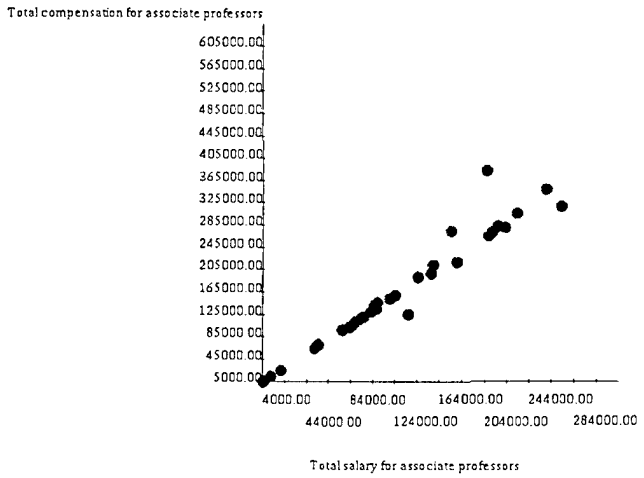


Figure C-8: Increasing trend

Another advanced find task is identifying group similarity, i.e. identifying a set of "similar" objects. The similarity attributes and the range of similarity, (i.e. what constitutes similar values) are often poorly defined, domain dependent, and cannot be easily captured in a task specification. The high articulatory distance therefore also makes these "find cluster" tasks more appropriate for perceptual processing.

In summary, advanced find tasks can be more effectively performed by mapping the source data to graphics because of the high articulatory load involved when using data computation functions. The role of data computation in this case is in directing the user's attention to objects that may be relevant. The system can make a best approximation of the find results and increase the saliency of those objects. Unlike the other data computation examples, however, we need to show all of the source data as well. This allows users to verify the system's results and to perceptually solve the task if the system approximation is incorrect or incomplete. For example in Figure C-8, the system may use data computation to approximate a line to the scatter points but still show the original data points so that users have the flexibility of verifying the system approximated line. Currently we do not deal with advanced find tasks in our work because of the high level of reasoning involved. Our work however does provide a basis that can be later expanded to deal with such tasks.

C-3.1.4 Comparison

Comparison tasks are used to determine relationships among data concepts. In this way, they are the complement of find tasks. In find tasks, the user knows the relationship to look for and is interested in the data concepts that participate in that relationship. On the other hand, in compare tasks, the user knows the set of data concepts to compare, but is unsure of the relationship(s) that exist among them. By comparing or analyzing different data attributes of the concepts, the user hopes to reveal the structure or relationships within the data.

We have found it useful to divide comparison tasks into three types: 1) comparing different data attributes within a single object set, 2) comparing the same data attribute across multiple different object sets, and 3) comparing different data attributes across different object sets. All three cases can be achieved with different combinations of logical *compare* and *lookup* operators.

1. Comparing different data attributes within a single object set

An example task of this type is comparing the *price-of-rice* and the *price-of-wheat* for particular *month* data concepts.

```
(Compare    (Lookup { month-1, month-2, …, month-n } ,
                    rice-selling-price ),
            (Lookup { month-1, month-2, …, month-n } ,
                    wheat-selling-price) )
```

In this compare task, we may be interested in several different aspects of the data. For example, we may be interested in value pair comparisons, i.e. seeing if the *rice-selling-price* is greater than the *wheat-selling-price* over *n* months or in a particular month. We may discover that "in *April*, the price of rice was much lower than the price of wheat due to excess production". Alternatively, we may be interested in group or trend comparisons. For example, we may want to compare the trend of rice prices over time with the trend of wheat prices over time. In this case we are more interested in gestalt results such as, "when the price of rice is increasing, the price of wheat tends to increase as well". Group comparisons are often used in this context to determine correlations between different data value sets.

2. Comparing the same data attribute across multiple different object sets

The second type of compare task reasons about the same data attribute over different object sets. For example we may want to compare the size of houses in the *Shadyside* area with the size of houses in the *Squirrel Hill* area. In this case there are two object sets, one with a membership of all *Shadyside* houses and the other with a membership of all *Squirrel Hill* houses. In the task specification below, these object sets are determined using the *find* task. Both sets of objects are compared based on the same data attribute, namely *house-size*.

```
(Compare    (Lookup     (Find   =, Lookup(all-houses, neighborhood),
                                Shadyside),
                        house-size ),
            (Lookup     (Find   =, Lookup(all-houses, neighborhood),
                                Squirrel Hill),
                        house-size ) )
```

One of the main differences between *type-2* compare tasks and *type-1* compare tasks is that *type-2* compare tasks may be used to compare value sets of different sizes. On the other hand, *type-1* compare tasks always operate on value sets that are of the same size because the values are extracted from the same data concepts. In addition, *type-1* compare tasks may be applied to attributes with different properties, for example to *house-price* which is a *quantitative* attribute and to *date-sold* which is a *temporal* attribute. This is however not true of *type-2* comparisons that are by definition applied to the same data attribute.

3. *Comparing different data attributes across different object sets*

The final type of comparison task involves the comparison of different data attributes across different data sets based on a common shared attribute. For example we may want to determine whether the standard of living (measured based on household income) in a particular neighborhood affects the selling price of houses. In this case, we are comparing the *income* attribute of *household* data concepts with the *selling-price* attribute of *house* data concepts based on a common attribute, *neighborhood.*

Type-3 comparisons can be transformed into *type-1* comparisons with some data adjustment. Specifically, we must reorganize the data so that the "*data object*" or data record represents the common comparison attribute. In the example presented above, we want to structure the data so that each data concept represents a different *neighborhood* (e.g. *Shadyside, Squirrel Hill, Pt.Breeze*) rather than an individual house (e.g. *634 Maryland Av.*). Each *neighborhood* data concept will then have an *average-household-income* attribute and an *average-house-price* attribute. The *type-3* comparison task will then reduce to a *type-1* comparison task as is shown below. These data restructuring operations always occur through data transformation functions.

```
(Compare    (Lookup { neighborhood-1, …, neighborhood-n } ,
                average-household-income ) ,
            (Lookup { neighborhood-1, …, neighborhood-n } ,
                average-house-price ) )
```

C-3.2 Task Extensions

The task model presented above is adapted from previous characterizations of data analysis tasks. Unfortunately, this task model is not sufficient to capture all the information necessary to effectively reason about data computation techniques and mapping techniques. In order to facilitate decision making between these two design alternatives we enrich our data analysis task model from previous approaches in three primary areas: task embeddings, task iteration, and task accuracy or precision.

C-3.2.1 Task Embeddings

Previous task models only allowed for 'flat' task specifications. Tasks are declared in isolation and task dependencies can only be captured with conditionals (e.g. Task C-1). Below (Task C-1), we show how Casner used if-then conditionals to specify that *downTime* should only be computed for the flight if the arrival time of the flight (*arrival1*) is before the start of the meeting (**beginMEETING).** This level of task dependency is insufficient for our work.

```
if (arrivesBeforeMeeting? arrival1 beginMEETING)
 then (computeDownTime beginMEETING arrival1 DOWNTIME)
```

Task C-1: Subtask extracted from Casner's airline-scheduling task [*Casner, 1991*]

In order to reason about data computation operations, we must identify *intermediate tasks*, which are tasks used for generating intermediate results - useful in servicing the end goal but not useful in themselves. For example, consider the task specification below:

```
(Ratio   (Difference (Lookup (  { all-houses } , asking-price ),
                     (Lookup (  { all-houses } , selling-price) ),
         (Difference (Lookup (  { all-houses } , date-sold  ),
                     (Lookup (  { all-houses } , date-on-market) )
         )
```

In this task, we are trying to determine how house prices change the longer they stay on the market. The two *difference* tasks are intermediate tasks and they are related in this case because they are both embedded within the same *ratio* task. This is an important relationship to capture, and it has implications for both data computation as well as mapping decisions. Intermediate tasks, for example, are very appropriate for data computation operations because their results need not be shown to users and this allows for significant data simplification (refer to chapter IV-3.2). In addition, task embeddedness indicates a closer relationship among all the data attributes involved and we can use this information to constrain all the data attributes so that they are mapped in closer proximity (e.g. to the same graphical object).

A task can be embedded within another if its output argument class corresponds to the input argument class of the other task. For example, we can embed a *compute* task within other *compute* tasks because the *compute* operation both accepts and produces *data values*. Similarly, we could also embed a *find* task within a *lookup* task, or a *compare* task within a *find* task. It is important to note that task `embeddedness' is just one of many possible relationships that can exist among tasks. Tasks may also be related because one is conditioned upon another (as in Task C-1), because they are applied to the same object sets or because one task is generated based on the processing of another task. For example, if we want to compute the *ratio* task shown above with data computation, we must generate a lookup task so users may view the results of the computed *ratio* operation.

C-3.2.2 Task iteration

Previous work dealt with task repetitions by constructing loops around the task structures [*Casner, 1991*]. Unfortunately, these loop structures are not interpreted by the automatic visualization designer. It is crucial to capture task iteration information in our work because it has significant implications for making data computation and mapping decisions. For example as was shown in chapter IV-3.4, *all-to-all* task repetitions (i.e. processing each and every value in a set with all values in a second set) behave very differently from *value-pair* task repetitions (i.e. processing each value in a set with its corresponding value in a second set). The first is usually more appropriately solved with mapping designs whereas the second is more appropriately solved with data computation.

We deal with task repetitions by applying a task to the entire set of objects that we want it to iterate over and then specifying the type of task repetition that we desire. There are currently three classes of task repetition options: *value-pair* (*one-to-one*), *all-to-all*, and *previous-pair*. To illustrate each repetition class we apply them in turn to the two sets of values shown below.

{ A, B, C, D, E }

{ 1, 2, 3, 4, 5 }

Value-pair tasks are repeated over each pair of corresponding values in the two data value sets (i.e. they are applied to (A, 1) , (B, 2), (C, 3), (D, 4), and (E, 5)). *All-to-all* tasks on the other hand require each value in the first set to be paired with all values in the second set (i.e. they are applied to (A, 1), (A, 2), ... , (A, 5), (B, 1), (B, 2), ..., (B, 5), (C, 1), (C, 2), ...). Finally, *previous-pair* operations ranks the input value set based on an ordering attribute and an ordering function. The task is then applied to each consecutive value pair in the ranked value set. For example, suppose the second value set is the ordering attribute for the first value set. Further suppose that the ordering function is the *greater-than* (>) operator. In this case the first value set will be ordered as follows { E, D, C, B, A } and the pairwise comparisons will be between (E, D), (D, C), (C, B), and (B, A). *Previous-pair* iterations are commonly applied to determine changes in a data attribute (e.g. *amount-of-sales*) with respect to changes in an independent ordered attribute (e.g. *time*). For example, we can use the *previous-pair* iteration to determine how the sales of a particular company are changing with time.

C-3.2.3 Task precision

Previous automatic visualization design systems use task accuracy as the most important effectiveness criteria for making data to graphical mapping choices. However, accuracy or precision can sometimes only be attained at the cost of significant cognitive load as we had shown in chapter IV-3.1. Depending on the task, it is not always necessary to attain high accuracy levels. For example, in some cases it may be sufficient that we know the net profit for month 1 is approximately 2 or 3 rather than exactly 1.6. If a system knows how important *accuracy* is to the user, it can make better choices between the many possible design alternatives.

For simplicity reasons we currently allow for three levels of accuracy: *high-accuracy*, *normal-accuracy*, and *fuzzy-accuracy*. High accuracy requires exact precision. For example in the difference task above, we want the differences calculated to all possible decimal values. Normal accuracy allows for an approximate comparison judgement. Relative values may be compared with no guarantees on the maximum amount of error. Finally *fuzzy-accuracy* refers to the case where we explicitly do not want accuracy. Here, we are specifying to the system that the task constraints are not absolute and that they should be relaxed. The default accuracy is *normal-accuracy*. There is a much wider range of accuracy levels than the ones that we have provided. However, we believe that our three accuracy levels covers many of the accuracy issues that arise when deciding between data computation and mapping operations. A

deeper treatment of task *accuracy* is left for future work. The idea of attaching precision levels to tasks is not entirely new. In AutoVisual [*Feiner, 1990*], the degree of precision may be specified for the object selection task, which corresponds on some levels to our *find* task.

In the next section, we consider how variations in the tasks described in this section can influence design choices between data computation and mapping operations. This analysis is accompanied with designs generated by our prototype automatic visualization design system as well as discussions on why certain designs were ranked higher based on the dimensions described in chapter IV-2.

C-4 Exploring the Space of Data Techniques and Mapping Techniques

In this section, we describe how data computation vs. mapping design decisions are made in our automatic design system for the range of goals in our task model. We generate the possible range of goals by varying the task specification in three primary ways:

1. *Task structure variation*: In section C-3, we described four task classes in our framework, *lookup, compute, compare* and *find*. We may change the task structure by using different task classes or different task operators within each class. For example, for a *compute* task, we may be interested in the *addition, difference, multiplication,* or *quotient* operators. The task structure can also be changed by reordering the operators in the task or by changing their embedding structure.
2. *Task input argument variation*: In section C-3.1 we show that tasks may be applied to attributes, objects, or values. Task arguments can also be left unspecified ("?") to indicate that an argument is unknown or that there are many possible argument alternatives.
3. *Task data sets*: Tasks may also be applied to different data sets. The data set size and distribution of values can also affect design decisions between data and mapping techniques.

In the following examples, we systematically generate a set of visualization designs for each goal using our automatic visualization design system (AVID). We then analyze the designs using the metrics in chapter IV-2 and discuss how they are ranked and which design guidelines are used in their ranking. In order to keep this section at a reasonable length, however, we will not present all of the visualization designs that are generated by our system. Instead, we only show those designs that have interesting differences. Note that the examples addressed in this section are purposely chosen for their simplicity so that we may highlight important design decision points and show them in isolation. In section C-5, we explore a more realistic task of purchasing a car.

C-4.1 Task Structure Variation

In this section, we vary the task structure but keep the task arguments and the data sets constant. There are three ways in which the task structure may be varied: 1) task class or operator variation, 2) task expansion, and 3) embedding structure variation. We first present an example task, which we will subsequently alter based on these three structural variations. For each variation, we consider its effects on data computation and mapping design decisions.

C-4.1.1 University Example

We are considering attending a university for undergraduate study, but we are concerned about financing issues. Thus, we want to view the combined *tuition* and *room & board costs* of our set of candidate universities. The specification for this task is shown below.

```
(Compute    Addition,
            (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
                    out-of-state-tuition),
            (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory}},
                    room-&-board-costs)  )
```

Task C-2: Computing the total cost for attending a university

Figure C-9 shows a data computation design and a mapping design for solving Task C-2. The data computation design (Figure C-9a) shows the pre-computed total costs (*out-of-state-tuition* + *room-&-board-cost*) for each university in a single column of figures and the mapping design (Figure C-9b) shows each of the cost figures separately in two different columns. In the data computation design users only need to look up a single total cost figure for each university, however, with the mapping design users must look up two cost figures and then perform the addition task cognitively. In this case, the data computation design is clearly much more effective than the mapping design because while the mapping design requires $2n$ perceptual lookups and n cognitive operations, the data computation design only requires n perceptual lookups. Our designer ranks Figure C-9a above Figure C-9b because it recognizes that the *addition* task cannot be effectively performed with text labels. Encoding the data with labels do not allow us off-load the addition task onto our perceptual system and as a result we must perform the task performed cognitively which has a high observational distance.

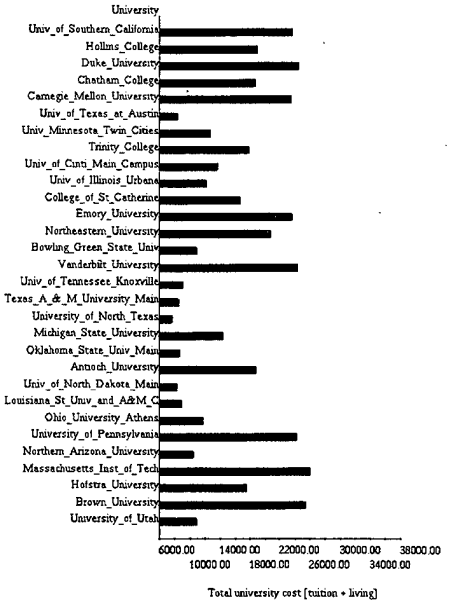
University	Total university cost [tuition + living]
Univ_of_Southern_California	23712.00
Hollins_College	18985.00
Duke_University	24540.00
Chatham_College	18730.00
Carnegie_Mellon_University	23590.00
Univ_of_Texas_at_Austin	8439.00
Univ_Minnesota_Twin_Cities	12693.00
Trinity_College	17842.00
Univ_of_Cinti_Main_Campus	13604.00
Univ_of_Illinois_Urbana	12134.00
College_of_St_Catherine	16664.00
Emory_University	23600.00
Northeastern_University	20805.00
Bowling_Green_State_Univ	10804.00
Vanderbilt_University	24390.00
Univ_of_Tennessee_Knoxville	9026.00
Texas_A_&_M_University_Main	8542.00
University_of_North_Texas	7683.00
Michigan_State_University	14392.00
Oklahoma_State_Univ_Main	8680.00
Antioch_University	18812.00
Univ_of_North_Dakota_Main	8337.00
Louisiana_St_Univ_and_A&M_C	8905.00
Ohio_University_Athens	11724.00
University_of_Pennsylvania	24290.00
Northern_Arizona_University	10474.00
Massachusetts_Inst_of_Tech	26075.00
Hofstra_University	17520.00
Brown_University	25454.00
University_of_Utah	10832.00

(a) Data computation design
Out-of-state-tuition **and** ***room-&-board-costs*** **are pre-added with data computation operators and the results are shown as** ***text*** **in the table.**

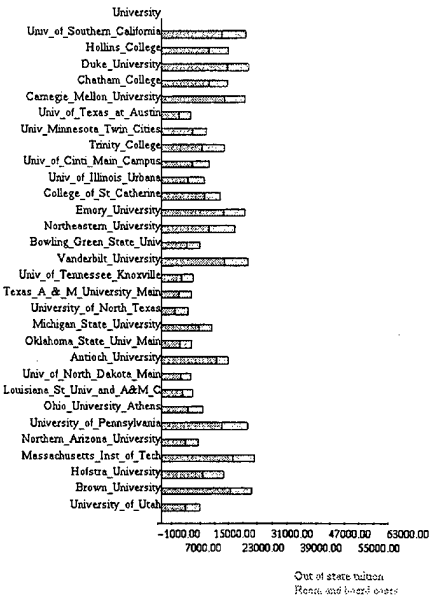
University	Room and board costs	Out of state tuition
Univ_of_Southern_California	6482.00	17230.00
Hollins_College	5515.00	13470.00
Duke_University	5950.00	18590.00
Chatham_College	5230.00	13500.00
Carnegie_Mellon_University	5690.00	17900.00
Univ_of_Texas_at_Austin	3309.00	5130.00
Univ_Minnesota_Twin_Cities	3744.00	6949.00
Trinity_College	6430.00	11412.00
Univ_of_Cinti_Main_Campus	4697.00	8907.00
Univ_of_Illinois_Urbana	4574.00	7560.00
College_of_St_Catherine	4440.00	12224.00
Emory_University	6000.00	17600.00
Northeastern_University	7425.00	13380.00
Bowling_Green_State_Univ	3352.00	7452.00
Vanderbilt_University	6525.00	17865.00
Univ_of_Tennessee_Knoxville	3262.00	5764.00
Texas_A_&_M_University_Main	3412.00	5130.00
University_of_North_Texas	3579.00	4104.00
Michigan_State_University	3734.00	10658.00
Oklahoma_State_Univ_Main	3344.00	5336.00
Antioch_University	3336.00	15476.00
Univ_of_North_Dakota_Main	2703.00	5634.00
Louisiana_St_Univ_and_A&M_C	2980.00	5925.00
Ohio_University_Athens	4095.00	7629.00
University_of_Pennsylvania	7270.00	17020.00
Northern_Arizona_University	3728.00	6746.00
Massachusetts_Inst_of_Tech	5975.00	20100.00
Hofstra_University	5920.00	11600.00
Brown_University	5926.00	19528.00
University_of_Utah	3975.00	6857.00

(b) Mapping design
Room-&-board-costs **is shown as** ***text*** **in the left table and** ***out-of-state-tuition*** **is shown as** ***text*** **in the right chart. This design solution is very inefficient because users must perform the addition task cognitively.**

Figure C-9: Design solutions for total cost computation task (Task C-2).



(a) Data computation design
Out-of-state-tuition **and** ***room-&-board-costs*** **are pre-added with data computation operators and the results are encoded on the** ***x-axis*** **of the bar-chart.**



(b) Mapping design
Out-of-state-tuition **is mapped to the** ***x-length*** **of the red bars and** ***room-&-board-costs*** **is mapped to the** ***x-length*** **of the blue bars. Total cost can be easily determined perceptually by looking at the combined lengths of the stacked bars.**

Figure C-10: Encoding the same data as Figure C-9 but with bars instead of text

A more effective mapping design, however, would offload the addition cognitive operation onto the user's perceptual system by mapping the two cost figures to bar lengths instead of text. Figure C-10 shows the same data as Figure C-9 except that we use bars instead of text to encode the data. The data computation design shows the total cost on the *x-axis* and the mapping design shows *out-of-state-tuition-cost* and *room-&-board-cost* on the *x-axis* using two stacked bars. In this example, the mapping design is significantly improved over the textual design in Figure C-9b. In fact both data and mapping designs in Figure C-10 take the same number of perceptual lookups. I.e. Figure C-10b and Figure C-10a are ranked at the same level by our automatic system. This example shows that the difference in effectiveness between a mapping and data computation design can change significantly depending on the graphical properties used to encode the data. In the subsequent examples, we will only compare data computation and mapping designs that use the best possible graphical properties.

Our automatic visualization design system however must be able to reason about both graphical property effectiveness together with functional effectiveness (i.e. data or mapping technique effectiveness) and this can sometimes be difficult. For example it can be difficult to decide whether the textual data computation design (Figure C-9a) is better than the bar mapping design (Figure C-10b) because both designs utilize different graphical property encodings as well as different design functions (i.e. data computation vs. mapping functions). The more appropriate design here can be based on task requirements, user preferences, domain conventions, etc. Some of these issues (e.g. task requirements) are taken into account in our system. The beauty of automatic visualization design however, is that it is a cooperative process between the user and the system. The advantage of such systems as a design tool is that it can quickly generate both Figure C-9a and Figure C-10b and show them to the user who can then decide between similarly effective designs based on their preferences.

C-4.1.2 Task Operator Variation

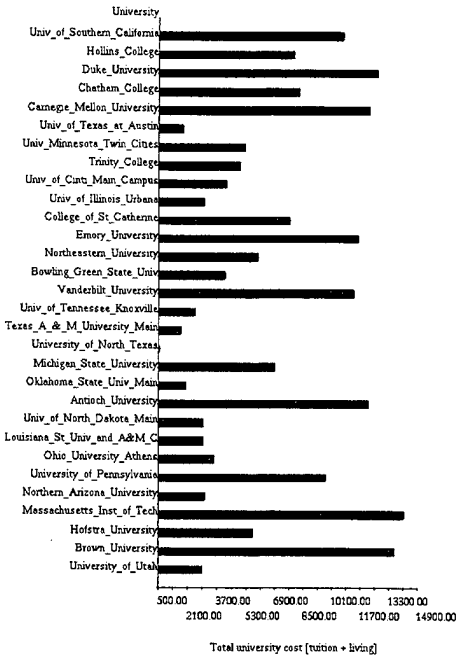
Now suppose that instead of computing the total cost for each university, we also want to determine whether the university charges (i.e. tuition) are in line with the living costs in the area. Specifically, we want to view the differences between *tuition-cost* and *room-&-board-cost* for each university.

```
(Compute    Difference,
            (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
                    out-of-state-tuition),
            (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
                    room-&-board-costs)  )
```

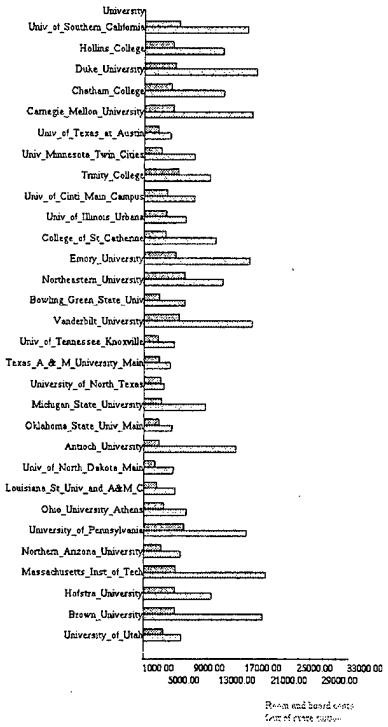
Task C-3: Change in task operator from *addition* to *difference*

In this task, the observational distance for the data computation design (Figure C-11a) consists of n perceptual bar length lookups. The mapping design (Figure C-11b) however requires a comparison between the *tuition-cost* bar and the *room-&-board-cost* bar, which results in $2n$ perceptual lookups. Thus unlike the

previous *addition* example, the data computation design is ranked above the mapping design in this example. This is because the graphical language available for showing the *difference* task (bar pairs) is less effective than the graphical language available for the addition task (stacked bars) (*availability of perceptual operators* guideline in chapter IV-3.3).



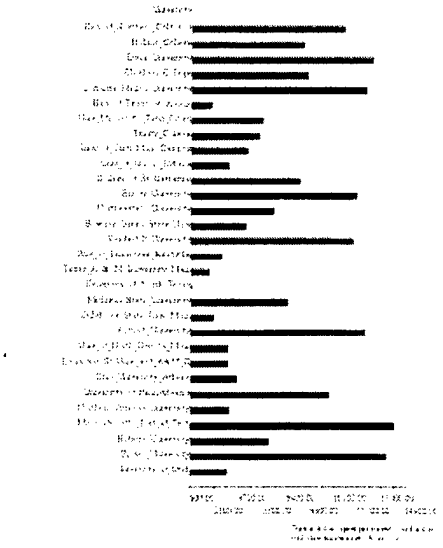
(a) Data computation design
The difference between *out-of-state-tuition-costs* and *room-&-board-costs* is pre-computed and encoded on the *x-axis* of the bar chart



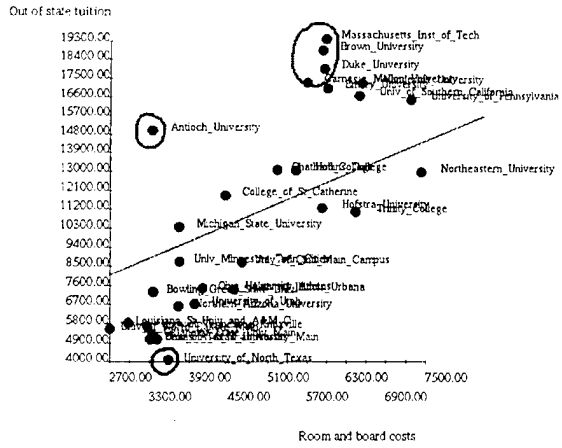
(b) Mapping design
***Room-&-board-costs* is mapped to the *x-length* of the red bars and *out-of-state-tuition* is mapped to the *x-length* of the blue bars. The cost differences can be perceptually determined by looking at the difference in lengths between each pair of bars. This perceptual design is less effective than the data computation design because the difference estimation operation is non-trivial unlike the addition operation in** Figure C-10(b).

Figure C-11: Computing the difference between *out-of-state-tuition* and *room-&-board-costs*

Now suppose that instead of computing the difference between *out-of-state-tuition-cost* and *room-&-board-cost*, we wanted to compute their ratio instead. Specifically, we want to see for each dollar spent on *room-&-board* how many dollars are spent on *tuition*. Figure C-12 shows the design alternatives for this task. In the data computation design the ratio values are pre-computed and mapped to the *x-lengths* of the bars. In the mapping design, the ratio values must be perceptual derived from the position of the marks which shows both *out-of-state-tuition-cost* on the *y-axis* and *room-&-board-cost* on the *x-axis*. To express the *ratio* task in the mapping design, our designer constrained both *cost* attributes to be mapped to *conjoint* properties on the same graphical object.



(a) Data computation design
The ratio of *out-of-state-tuition* dollars spent per *room-&-board-costs* dollar is pre-computed and shown on the *x-axis* of the bar chart. *University of North Texas* has a low ratio value while *MIT, Brown, Duke, CMU,* and *Antioch Universities* have high ratio values (i.e. tuition costs are high wrt. living costs).



(b) Mapping design
***Out-of-state-tuition* is mapped to the *y-axis* and *room-&-board-costs* is mapped to the *x-axis*. To estimate the ratio values between *out-of-state-tuition* and *room-&-board-costs* we look at the distance of the points from the average ratio line (shown in red). The points to the top that are circled red (*MIT, Brown, Duke, CMU, Antioch*) are the ones with high ratio values while the points to the bottom are the ones with below average ratio values (*University of North Texas*).**

Figure C-12: Computing the ratio between *out-of-state-tuition-costs* and *room-&-board-costs*

The mapping design (Figure C-12b) is not very effective if we want to determine exact ratio values. However, our intention is to find outlier universities, i.e. universities with either inordinately high or low *tuition-cost/room-&-board-cost* ratios then Figure C-12b is a useful solution. For this task, high accuracy is not required because we are only interested in relative ratio values. In fact, the perceptual load for both the data computation and mapping designs are similar (ignoring the occlusion problem). For example in both designs we can pre-attentively see that *Antioch University, MIT, Brown, Duke,* and *CMU* (circled in *red*) have unusually high *tuition/room-&-board-cost* ratio while *University of North Texas* (circled in *blue*) has an unusually low *tuition/room-&-board-cost* ratio. Our automatic designer ranked the data computation design slightly above the mapping design because of the greater perceptual complexity (i.e. graphical density) in the mapping design.

C-4.1.3 Task Expansion

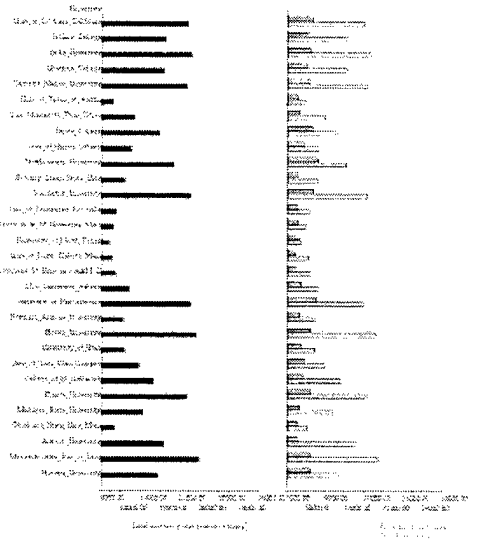
In addition to being interested in the total cost for attending each university, suppose we are also interested in the individual *tuition-cost* and *room-&-board-cost* figures.

```
(Compute     Addition,
             (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
                     out-of-state-tuition),
             (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
                     room-&-board-costs)  )
(Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
          out-of-state-tuition),
(Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
          room-&-board-costs)
```

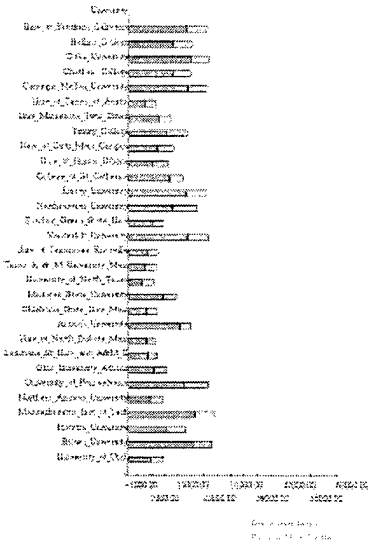
Task C-4: Expanded university total-cost task

Specifically, we want to see if higher cost universities are a result of additional *tuition-cost* or *room-&-board costs*. Thus, we have expanded the task specification from Task C-2 with additional *lookup* tasks.

Unlike the original task (Task C-2), our designer picked the mapping design (Figure C-13b) as the most effective for this expanded task. The mapping design is chosen based on the *task variation on attribute* design guideline in chapter IV-3.5 because it allows us to solve the *addition* and subsequent *look up* tasks by using the same set of graphical objects. All the task information is presented in a space efficient manner, and the observational distance for performing the *addition* task in the mapping design is negligible because *stacked bars* are very effective for addressing *addition* operations. On the other hand, the data computation design (Figure C-13a) has a greater number of graphical objects (two separate charts and three bars) because we must show both the computed *total cost* results as well as the original *tuition-cost* and *room-&-board-cost* attribute values.



(a) Data computation design
In this example we want to see both total costs (*out-of-state-tuition* + *room-&-board-costs*) as well as the individual costs. In this design total costs are pre-computed and shown on the *x-axis* of the left chart. The individual cost values are shown as blue and red bars respectively in the right chart.



(b) Mapping design
***Out-of-state-tuition* is mapped to the *x-length* of the red bar and *room-&-board-costs* is mapped to the *x-length* of the blue bar. This design is very effective because it allows us to easily derive the total cost values by looking at the combined lengths of the stacked bars. However, the two individual cost values are also readily accessible.**

Figure C-13: Design solutions for expanded total cost computation task (Task C-4)

Now, instead of computing the total cost for each university, suppose we are curious to see if there are any universities with *tuition-cost* less than *room-&-board-cost*.

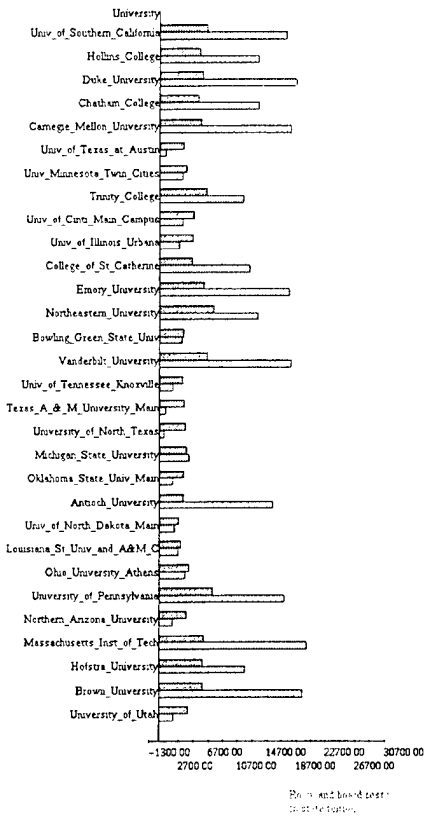
```
(Find    <,
         (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
                 in-state-tuition),
         (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
                 room-&-board-costs)  )
(Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
        in-state-tuition)
(Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
        room-&-board-costs)
```

Task C-5: Expanded tuition-cost and room-&-board-cost find task

Not surprisingly there are no universities with *out-of-state-tuition-cost* less than *room-&-board-cost* however there are some universities with *in-state-tuition-cost* less than *room-&-board-cost* as is shown in Figure C-14.



(a) Data computation design
In this design universities with *in-state-tuition* less then *room-&-board-costs* are shown in red. Not surprisingly these tend to be state universities. The individual *room-&-board-costs* (top bar) and *in-state-tuition* (bottom bar) values are encoded on the *x-axis*. Unlike Figure C-13a, **the computed results can be more effectively integrated into the design here, making it more effective compared to the mapping design.**



(b) Mapping design
***Room-&-board-costs* is mapped to the *x-length* of the red bars and *in-state-tuition* is mapped to the *x-length* of the blue bars. To find the universities with greater *room-&-board-costs* compared to *in-state-tuition* we must compare the lengths of each bar pair which is a fairly time consuming operation.**

Figure C-14: Design solutions for expanded find task (Task C-5)

Unlike the previous expanded task (Task C-4), the most effective design here is the data computation design (Figure C-14a). In Figure C-14a, the *find* task is performed with data computation functions and its results are shown using *color*. The blue hue bars show all the universities with *in-state-tuition-cost* less than *room-&-board-costs*. This design is effective because the *find* results can be introduced into the graphic without adding much visual complexity and without cluttering up the display space (in contrast to the data computation design in Figure C-13a). In addition, perceptual load has been reduced significantly because the find results can be read in a single pre-attentive perceptual operation (i.e. perceptual load = p). The mapping design, however, requires users to scan through all bars to visually search for all instances where *tuition-cost* is less than *room-&-board-cost* (i.e. perceptual load = np where n is the number of universities). The data computation design also presents the find results with great accuracy, leaving no room for perceptual errors.

This example illustrates the interaction between two different design guidelines, the *availability-of-perceptual-operator* guideline and the *task-variation* guideline. In this example, the available perceptual operator for performing the *find* task is not as effective as using data computation operations (i.e. preference for data computation design). However, there is *task-variation* on the *find* task and this usually results in greater perceptual complexity for the data computation design (i.e. preference for mapping design). While perceptual complexity in Figure C-14a is larger than in Figure C-14b because of the additional *color* encoding, the added complexity here is relatively small. Only a single graphical property is added here compared to the additional graphical object and region added in Figure C-13a. Consequently, the perceptual savings enabled for the *find* task through data computation outweighs the small added complexity from the use of *color*. Our automatic designer deals with interacting guidelines by adding costs to each design alternative based on the guideline being violated, and the depth of the violation. Specifically, a higher cost is added to Figure C-13a for the *task-variation* guideline violation compared to Figure C-14a, because in the former case, the added perceptual complexity is more significant.

C-4.1.4 Embedding Structure Variation

Since universities with *tuition-cost* less than the *room-&-board-cost* are such anomalies, we might want to examine the individual *tuition-cost* figures for only those universities to see which university has a highest and lowest costs within the set. This task is very similar to Task C-5 except that the embedding structure is changed. Specifically, the *find* task is now embedded within the two lookup tasks to indicate that we **only** want to lookup the cost values for the anomalous universities.

```
(Lookup (Find    <,
                 (Lookup { ... }, in-state-tuition),
                 (Lookup { ... }, room-&-board-costs)      )
        in-state-tuition)
```

Task C-6: Change in task embedding structure from Task C-5
(Note that we represent the university set with { ... } here to make the task specification easier to read. However, the data set used is the same as all previous examples in this section).

In this case, the most effective design (Figure C-15) performs the find task with data computation, and only shows *in-state-tuition-cost* figures for those universities with greater *room-&-board-cost* compared to *in-state-tuition-cost*. All other universities are culled from the display based on our *intermediate task* design guideline (chapter IV-3.2) which calls for all intermediate results to be hidden in order to reduce display complexity.

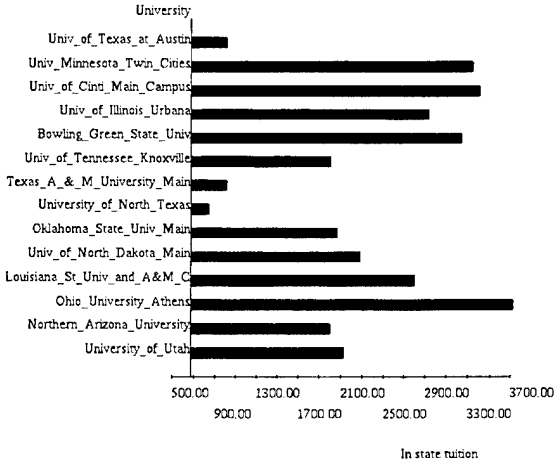


Figure C-15: Data computation design for Task C-6
In this design the automatic design system is able to utilize the embedding structure of the task to filter the design so that only those universities with *in-state-tuition* less than *room-&-board-costs* are shown. This cuts down on the number of elements that have to be shown significantly, producing a more easily interpretable display. *In-state-tuition* values are then shown on the *x-axis* of the bar chart.

We can also solve Task C-6 using the designs in Figure C-14 (i.e. our automatic designer will also generate those designs as alternatives for Task C-6). However, they are ranked lower because they both have greater perceptual clutter (i.e. a greater number of graphical objects), require more display space, and require more perceptual processing compared to Figure C-15. In particular, Figure C-14a requires an additional perceptual operation for getting all the purple bars, and Figure C-14b requires n additional perceptual operations for comparing each bar pair (where n = number of universities). In this example, our automatic system is able to exploit the additional task information captured by the embedding structure of the task to generate a more effective design than what was possible in Task C-5 which has a flat, unembedded structure.

C-4.2 Task Argument Variation

As was shown in section C-3.1, the data analysis tasks we consider in this thesis can be applied to three argument types: *values, attributes,* and *objects*. Tasks may also accept additional input arguments such as *iteration type* and *task type* as was described in section C-3.2. In this section, we consider how different input task arguments may result in different data and mapping choices. Specifically, there are three classes of input argument variations:

1. *Use of a data attribute value set vs. use of constants*: Tasks may be applied to single values or to sets of values. When tasks are applied to sets of values, we may iterate over these values in a variety of ways (*value-pair, all-to-all, previous-pair*).

2. *Use of known vs. unknown arguments*: According to the task model described in section C-3.1, task arguments may be specified or left as an unknown (?). In the latter case, we are indicating that there are many possible task arguments and the proper one(s) for the task can only be determined during exploration.

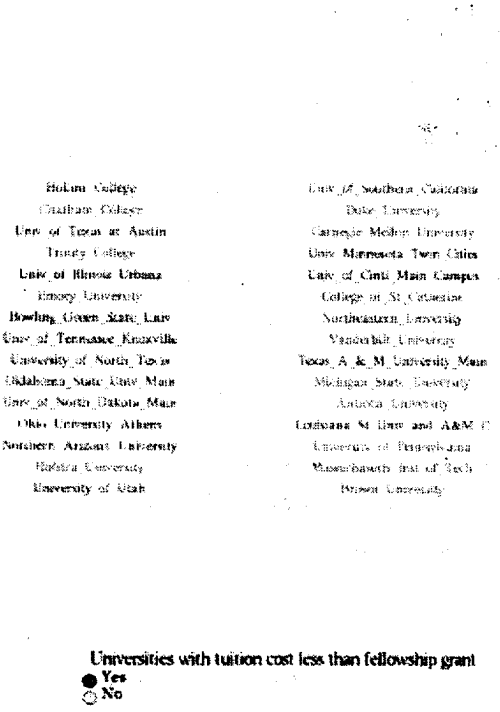
3. *Use of different classes of attribute types*: Data attributes are associated with a set of characterizations that describe their *set ordering* (quantitative, nominal, or ordinal), *domain of membership* (time, space, temperature, or mass), and relational structure [*Roth, 1990*; *Zhou, 1996*]. Previous work on automatic visualization has shown that these characterizations have significant impact on the choices made in mapping data to graphics. Here we consider some of the implications that these characterizations have on data vs. mapping decisions.

C-4.2.1 Use of constants vs. data attribute value sets

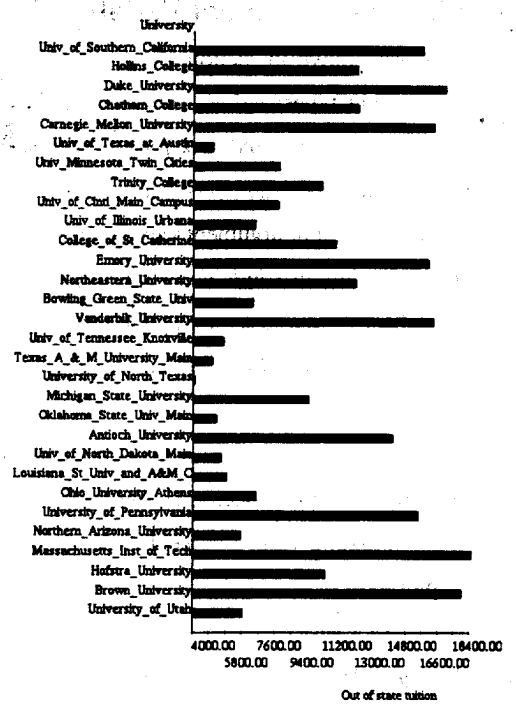
In this section, we continue our university-financing example. Suppose that we have received a fellowship that covers up to 10k worth of tuition cost. We might only want to attend universities with tuition costs below this figure so that we need not worry about paying any additional fees. The task specification is similar to Task C-5 except that instead of comparing two sets of attribute values (i.e. *tuition-cost* and *room-&-board-cost*), here we are comparing an attribute value set (*tuition-cost*) with a constant value (*10k*).

```
(Find    <,
         (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
                 out-of-state-tuition),
         10k   )
```

Task C-7: Find task with a constant argument



(a) Data computation design
Universities with *out-of-state-tuition* less than 10k are shown in red.



(b) Mapping design
***Out-of-state-tuition* is mapped to the *x-axis*. To find universities with *out-of-state-tuition* less than 10k we must look on the *x-axis* for the 10k point and then compare the bar lengths to this point. Even though this operation is more difficult than finding the red labels in design (a), we still have access to the actual *tuition* figures here, so we may compute the amount saved from the 10k mark. This information is lost in design (a).**

Figure C-16: Design solutions for find task in Task C-7

The data computation design (Figure C-16a) pre-computes the *find* task and maps the results to *hue*. The *red* hued universities are the universities that pass the *find* task and the *blue* hued universities are the ones that did not pass the *find* task. The mapping design (Figure C-16b) maps the *out-of-state-tuition-cost* values to a set of *bar-lengths* and leaves it to the user to pick the bars that exceed *10k* in value. The mapping design for this example (Figure C-16b) is much more effective than the mapping design for Task C-5 (Figure C-14b). This is because the perceptual operation for finding all bars less than a *constant-value-line* (Figure C-16b) is pre-attentive while the operation for finding all bar pairs with *tuition-cost* less than *room-&-board-cost* (Figure C-14b) is not. Thus, our automatic designer assigns a lower cost to the *mapping-find* design when one of the *find* task arguments is a constant value.

In this task the observational distance for both the data computation and mapping designs are comparable. Both solutions only require a single pre-attentive perceptual operation to find all universities that have greater *tuition-cost* than *10k*. However, our designer still ranks the data computation design ahead

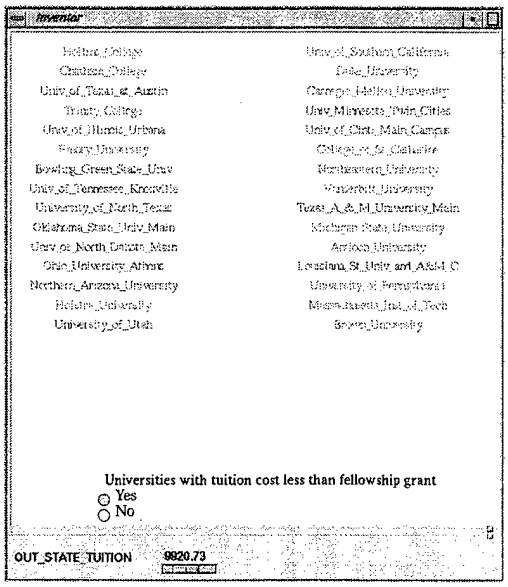
of the mapping design because it is more difficult to determine the *find* results with high *accuracy* using the mapping design. This is consistent with our *accuracy* heuristic in chapter IV-3.1.

C-4.2.2 Use of known vs. unknown arguments

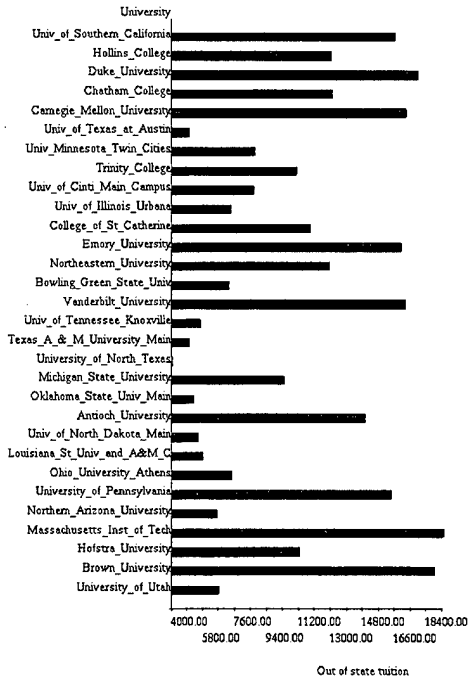
Now, suppose we are unsure of the amount covered by our fellowship because the figure is dependent on our expected *SAT* and *ACT* test scores. One way to perform this task is to leave the fellowship constant as an unknown ("?") in the specification.

```
(Find    <,
         (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
                 out-of-state-tuition),
         ? )
(Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
         out-of-state-tuition)
```

Task C-8: Find task with an unknown task argument



(a) Data computation design
This design is similar to Figure C-16a **except that here we must enter in the *out-of-state-tuition* threshold value we are interested in through a *slider* input device. This increases the overall load placed upon the user, making the mapping design alternative more attractive.**



(b) Mapping design
Exactly the same design can be used here as was used in Figure C-16b **even though we do not know the fellowship value in this case (i.e. task specificity is lower). This is because all of the data is presented to the user, thus there is greater flexibility in the range and types of questions that may be answered with this display.**

Figure C-17: Design alternatives for Task C-8

For tasks with unknown arguments, a mapping design is generally preferred over a data computed one because the task specificity is low (*task-specificity* guideline in chapter IV-3.6) and the articulatory distance is higher if we use data computation. The mapping design in Figure C-17b shows the *tuition-cost* figures as bar lengths and leaves it up to the user to determine the fellowship threshold line perceptually. The observational distance for this design is *x2p* to perform the *find* task, where *x* represents the number of times the *find* task is repeated, and *p* represents the load of a simple perceptual operation. Each time the *find* task is repeated, a total of *2p* perceptual load is required; *1p* load to determine the threshold line on the *x-axis* and another *1p* to pre-attentively identify all bars below or above that threshold.

The data computation design (Figure C-17a) provides a *slider* input device that allows users to manually enter in different fellowship values. Universities with *tuition-cost* below this input value are then highlighted pink. In this case, the perceptual load is slightly smaller. Only a single perceptual operation is needed for each task iteration to pre-attentively identify all the pink university names (i.e. load = *xp* where *x* represents the number of times the *find* task is repeated). However, in addition to the perceptual load there is also an articulatory load. Each time the *find* task is repeated, the user must enter a new input value through the slider. The load for a single input is *2(m+k)* i.e. two mouse moves (*2m*) with one mouse click and one mouse release (*2k*). Thus, the total articulatory load is *2x(m+k)*. This articulatory load outweighs the additional perceptual load needed in the mapping design thus our designer ranks the mapping design above the data computation design.

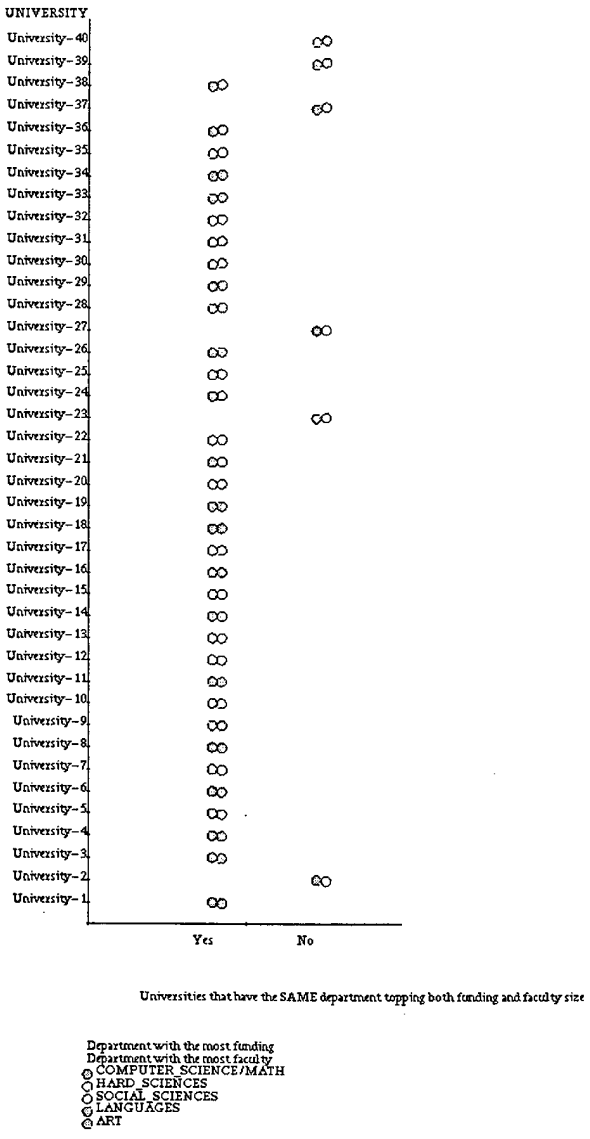
We want to point out however that accuracy is assumed to be less important for the *find* task here. When high accuracy is required, the data computation design becomes more effective because it can be difficult to get highly accurate find results from the mapping design. In particular, it may be difficult to determine which point on the *x-axis* corresponds to a desired fellowship figure and it may be difficult to perceptually project a straight line upwards from the *x-axis* to accurately process the *bar-lengths* especially for *tuition-cost* figures that are close in value to the fellowship. This is consistent with our *accuracy* design guideline in chapter IV-3.1.

C-4.2.3 Change in attribute or value type

In addition to financial costs constraints, suppose we are also only interested in finding out the departments in each school which are most prosperous (i.e. has the most funding as well as the most faculty). Of particular interest are the departments that top both the categories. Analyzing this data will help us determine which areas of study are the most popular and financially rewarding. Note that this task is identical to Task C-5 except that instead of comparing two quantitative attributes (*tuition-cost* and *room-&-board-cost*) we are comparing two nominal attributes (*department-with-most-funding* and *department-with-most-faculty*).

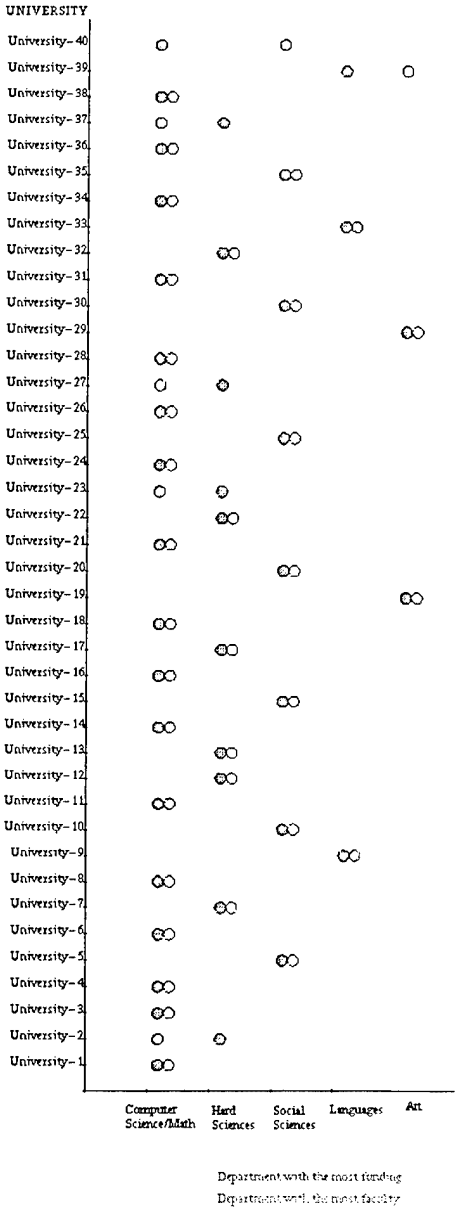
```
(Find    =,
         (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
                 department-with-most-funding),
         (Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
                 department-with-most-faculty)     )
(Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
         department-with-most-funding),
(Lookup {Carnegie_Mellon_University,MIT, Duke, …, Emory},
         department-with-most-faculty)
```

Task C-9: Finding the most prosperous department in each university based on funding and faculty size

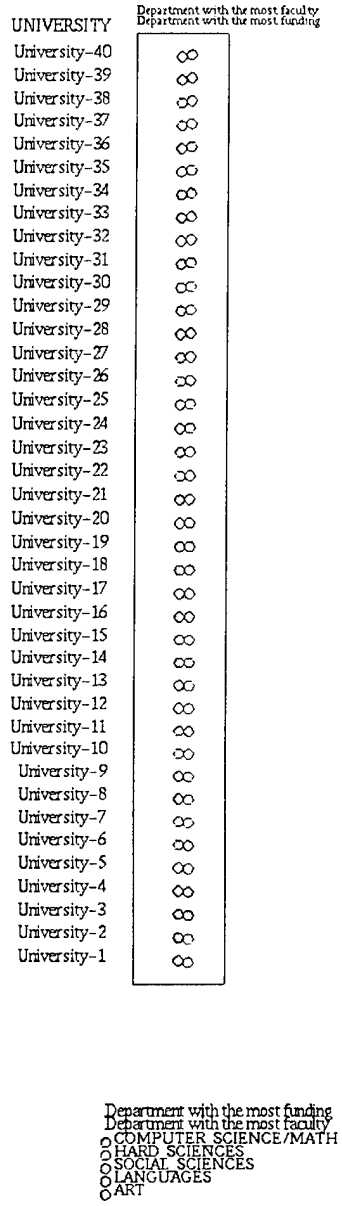


(a) Data computation design

Each university (*y-axis*) is associated with two marks, one representing the department with the most funding (left) and the other representing the department with the most faculty (right). The five main possible departments are represented using *hue*. In addition, universities with the same department topping both funding and faculty are pre-computed and shown in the left column.



(b) Mapping design with *position*
***Universities* are mapped to the *y-axis* and the five different departments are mapped to the *x-axis*. The department with the most funding is indicated with a red mark and the department with the most faculty is indicated with a blue mark. Universities with the same top department would have the two red/blue marks together. Finding such universities can be performed pre-attentively or semi pre-attentively with this display.**



(c) Mapping design with *hue*
***Universities* are mapped to the *y-axis* and the different department types are mapped to *hue*. Universities that have the same top faculty and funding departments are those with pair-marks that have the same color. This design is less effective than design (b) because there is a lot of *hue* noise here making it more difficult to identify identically colored marks.**

Figure C-18: Finding universities where the department with the most faculty is also the department with the most funding. The two mapping alternatives are less effective compared to the data computation solution because in both the mapping designs the *find* task may not be fully pre-attentive while in the data computation design it is very easy to identify the universities with the same top faculty and funding departments. In addition, this information is very well integrated into the design without increasing complexity by much.

In the data computation design, the universities that pass the *find* task are shown on the left and the universities that do not pass the *find* task are shown on the right. Each university is represented by a cluster of two marks, with the first mark representing the department with the most funding and the second mark representing the department with the most faculty. In this example, the data computation design is ranked higher than the mapping solutions because it allows pre-attentive performance of the *find* task, and the additional *find* results can be integrated with the two *lookups* very well, without significantly increasing graphical complexity.

On the other hand, the mapping solution does not ensure pre-attentive performance of the *find* task. Figure C-18b shows a mapping design where the *department* attribute is mapped to *x-position* and the department with the most funding is represented by a *pink mark* while the department with the most faculty is represented by a *blue mark*. We can find the universities that fulfill our task constraint by looking for clusters of two marks (i.e. a two mark cluster indicates a university that has the same department topping both faculty and funding). Figure C-18c shows a different mapping design alternative where the *department* information is mapped to *hue* instead of *x-position*. The left *marks* in Figure C-18c represent the departments with the most funding and the right *marks* represent the departments with the most faculty. Unlike the data computation solution (Figure C-18a), pre-attentive perception is not possible here because there are too many colors in the display and the noise from these colors make it difficult to pre-attentively identify rows that have the same colored *marks*.

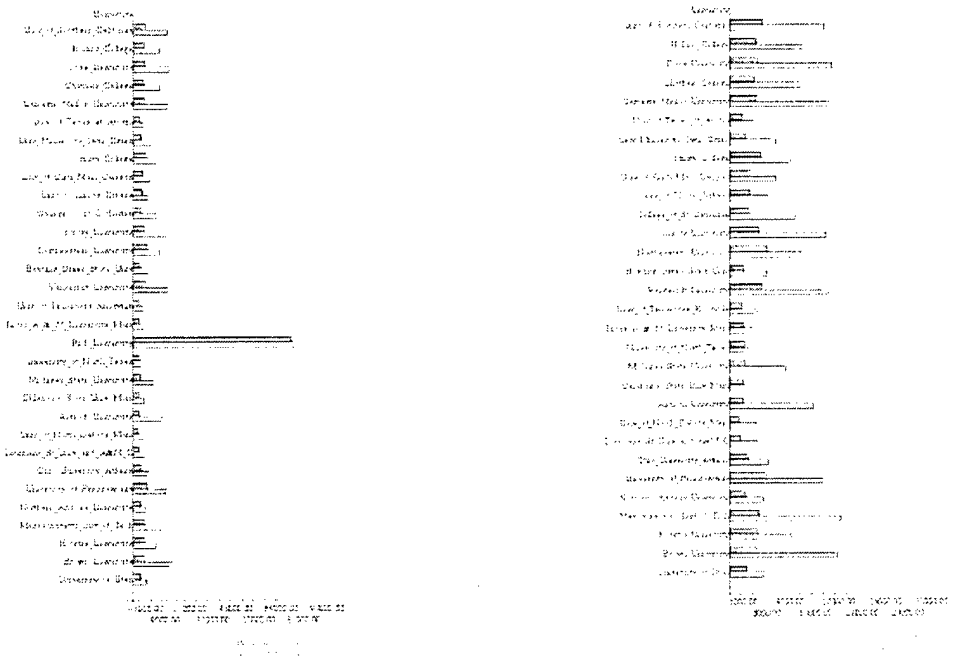
The important issue here, however, is that the mapping design for tasks with *nominal* attributes is more effective than the mapping design for tasks with *quantitative* attributes (e.g. in Task C-5). In the former case, the departments can be compared with 100% accuracy and the task can sometimes be accomplished pre-attentively. In the latter case, the mapping design is much less accurate and cannot be accomplished pre-attentively. Thus a lower cost should be assigned to mapping designs for tasks with *nominal* or *ordinal* input arguments compared to tasks with *quantitative* input arguments.

Nominal and ordinal data attributes are probably the easiest to process perceptually especially when they are well-bounded (i.e. they represent only a small number of different values or have a value membership set that is relatively small). Attribute values that are discrete (i.e. not continuous) and well-bounded can be to be encoded with a wide range of pre-attentive graphical properties that are maximally differentiated in value, which makes perceptual processing significantly easier and more accurate. Discrete attributes are less effective because although they are non-continuous, they are unbounded. Quantitative

values are the most difficult to process perceptually because they are continuous and unbounded. This makes it difficult to accurately translate the perceptual values back into data values[1].

C-4.3 Data Set Variation

Variation in the data set mainly causes readability problems (e.g. *occlusion, display density, dwarfing*) which increases both the expressive and observational distances of a graphic. Some of these problems may be avoided using data computation designs that can hide and summarize data, consequently reducing perceptual complexity and increasing readability. Two of the primary properties of a data set that can affect readability include: 1) the distribution of values within the data set, and 2) the data set size.

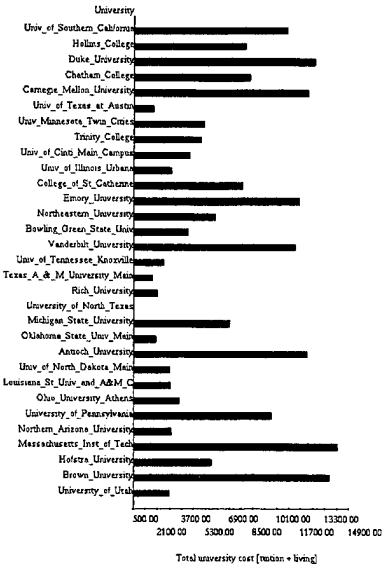


Data set with high cost *Rich-University* included

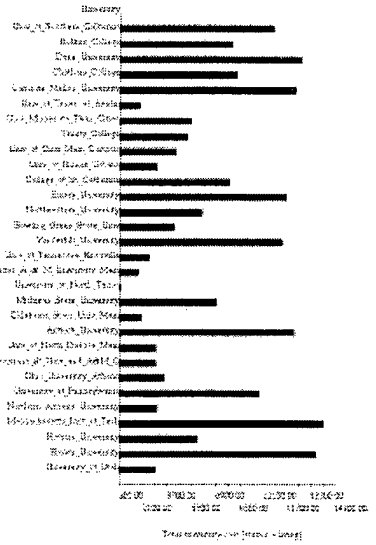
Data set with *Rich-University* removed

Figure C-19: Mapping design where *out-of-state-tuition-cost* is mapped to the blue bar *x-lengths* and *room-&-board-cost* is mapped to the red bar *x-lengths* for a set of universities. In the data set with *Rich-University* there are severe dwarfing problems on the bar lengths making it difficult to accurately estimate cost values.

[1] This statement is true for all graphical property encodings except for when *text-labeling* is used. However, *text* is not perceptually pre-attentive, thus it is not a very effective graphical property choice.



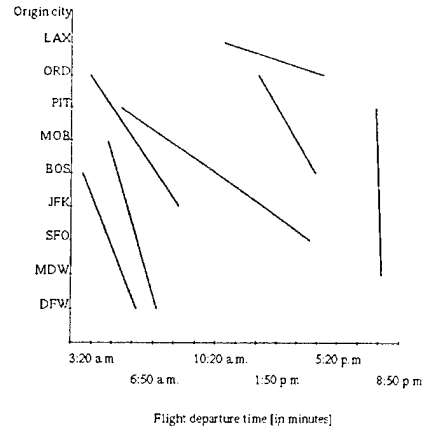
Data set with high cost *Rich-University* included (In this case *Rich-University* does not cause any dwarfing problems because we are only looking at the value differences rather than the actual cost values)



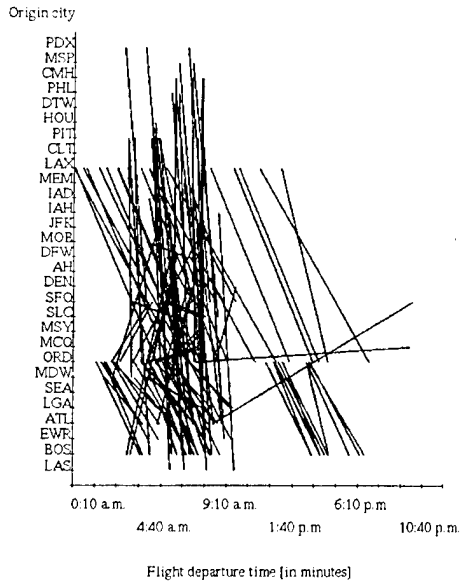
Data set with *Rich-University* removed

Figure C-20: Same data set as Figure C-19 but showing the pre-processed difference values between *out-of-state-tuition-cost* and *room-&-board-cost* instead of the original cost figures. Note that in this case it does not matter whether *Rich-university* is included or not, the difference distributions of the two data sets are comparable. I.e. the dwarfing problem is no longer an issue in the data transform design.

The distribution of values within a data set mainly affects the accuracy and complexity of achieving tasks. If a data set contains values that are widely set apart, then it is likely that the graphical scale used to encode the data will be much smaller than the data scale. This may cause some data values to be minimized or dwarfed so that it they are hard to access perceptually. For example, consider the *difference* task in Task C-2. Suppose that the costs of a particular university far exceed the others, as in Figure C-19-left. The long bars for *Rich-University,* cause the other cost bars to be greatly dwarfed. This makes perceptual judgements of bar length differences very difficult. On the other hand, if the distribution of values are close together, as in Figure C-19-right (which has the same data as the left figure but with the high cost university removed), the length differences can be determined with greater accuracy and with less probability for error. In cases where the data distribution results in intense data dwarfing, it is possible that using a data computation design will reduce the dwarfing effects. For example, Figure C-20 shows the pre-processed difference values of the data set in Figure C-19-left. In this design, the great dwarfing effect has been reduced because we are only showing the difference values between each bar pair instead of the values themselves.



(a) Smaller filtered airline data set that is much easier to process compared to design (b) because there is no occlusion here.



(b) Large unfiltered airline data set that is difficult to analyze because of the high degree of occlusion. With large data sets it is advantageous to do some pre-processing and pre-filtering to simplify the data set before mapping it to graphics or the resulting display will be difficult to analyze and process.

The size of the data set can also affect design choices. When data sets are large, data computation operators are usually preferable (to mapping operators) because they can summarize the data set and fewer elements need to be shown to users. For example, consider the airline-scheduling example shown below, where each flight is represented by a line. This graphic is appropriate when the flight database is small. However when there are many flights, we would quickly fill the visualization with so many lines that there would be too much occlusion to read anything useful from it. The data computation solution (Figure IV-3) however, does not have these problems because the data filtering possible significantly reduces the number of flights that need to be shown. The only exception is for *all-to-all* tasks (chapter IV-3.4) which should be performed with mapping techniques especially for larger data sets because the data computation solution significantly expands the amount of data in the visualization.

C-4.4 Summary

In this section we presented a set of simple examples and showed how variations to the 1) task, 2) task arguments, and 3) data sets, can result in different data computation and mapping designs. We intentionally used simpler examples so that the variations in design and user goals can be more clearly illustrated. We also briefly discussed how larger data sets and *all-to-all* task iterations can result in readability issues such as visualizations that are too dense, graphical elements that are too small or graphical elements that get dwarfed because of the wide distribution of values within the data set. As is shown in the examples here,

some of these readability problems can be avoided through making careful choices between data and mapping operators. However, this alone cannot solve all the readability problems that may arise. In appendix F we discuss how interactivity, graphical, and rendering transforms can also be used to alleviate many of these readability problems.

The designs shown in this section are all generated by our prototype automatic designer. Our designer uses a *cost* structure that is based on the design guidelines described in chapter IV-2, chapter IV-3 and on previous heuristics used in automatic data graphic design [*Mackinlay, 1986a, 1986b; Casner, 1991*]. This cost structure is used to make choices between different graphical artifacts and data functions. Designs that simplify the data or that have a lower semantic distance get a lower score, while designs with high complexity or a great semantic distance get a higher score (i.e. a lower cost or score indicates a better design). Designs are then generated according to increasing cost (i.e. decreasing effectiveness). Implementation details on our designer are described in chapter V. In the next section, we explore a more realistic task of purchasing a car and show how our designer may be used in an iterative data analysis session.

C-5 Purchasing a Car

In this example, Bob is planning to purchase a car. In picking a car Bob is interested in the following properties: 1) low fuel consumption (i.e. high miles per gallon), 2) cheap price, 3) high performance engine (i.e. high engine capacity), 4) good speed and power (i.e. high *horsepower*). It is obvious that Bob cannot get a car that has all of these properties because they conflict with one another. Higher performance cars are usually more expensive and cars with good fuel consumption (high mpg) usually have lower performance. Thus Bob would need to trade-off these properties with one another and pick the car with the "best" balance between price and performance. There are several possible ways to accomplish this task. One way is to simply determine the best car in each category, and pick the car that tops the most categories. The task specification is shown below:

```
Find ( = , Lookup( {all-cars}, price),
         Compute ( Min, Lookup( {all-cars}, price))
Find ( = , Lookup( {all-cars}, mpg),
         Compute ( Max, Lookup( {all-cars}, mpg))
Find ( = , Lookup( {all-cars}, engine-capacity),
         Compute ( Max, Lookup( {all-cars}, engine-capacity))
Find ( = , Lookup( {all-cars}, horsepower),
         Compute ( Max, Lookup( {all-cars}, horsepower))
```

Task C-10: Car purchasing task A

The most effective design chosen computes the *max, min* and *find* tasks internally and the user is only shown the cars that top each category (Figure C-21). A problem with specifying the task in this manner arises when each category is topped by a different car. When this occurs there is no way to pick the best candidate. In addition, even though a car may top one or multiple categories (e.g. *Metro* in Figure C-21) it may not necessarily be a good choice for Bob if its ranking is low in the other categories. This is because Bob is looking for a balance between both price and performance.

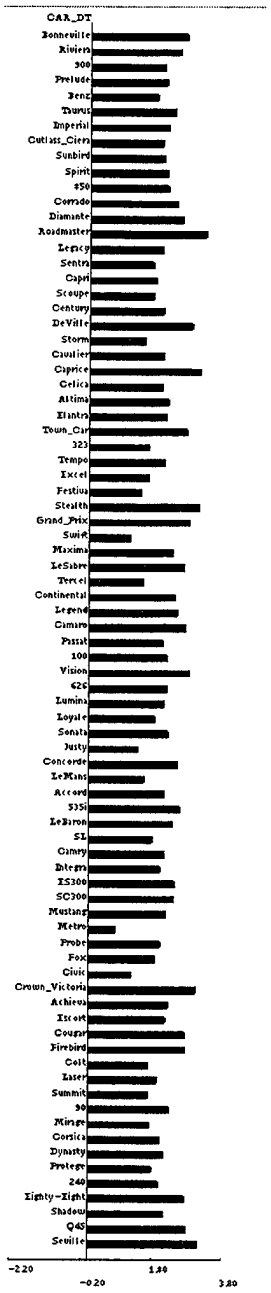
Horsepower	Engine-size	MPG in the city	Minimum price
Stealth	Roadmaster	Metro	Metro

Figure C-21: Data computation design for car purchasing task A (Task C-10)
The top cars are shown for each car picking category. While this design is very simple and easy to interpret it does not allow us to flexibly take all four car picking attributes into account simultaneously.

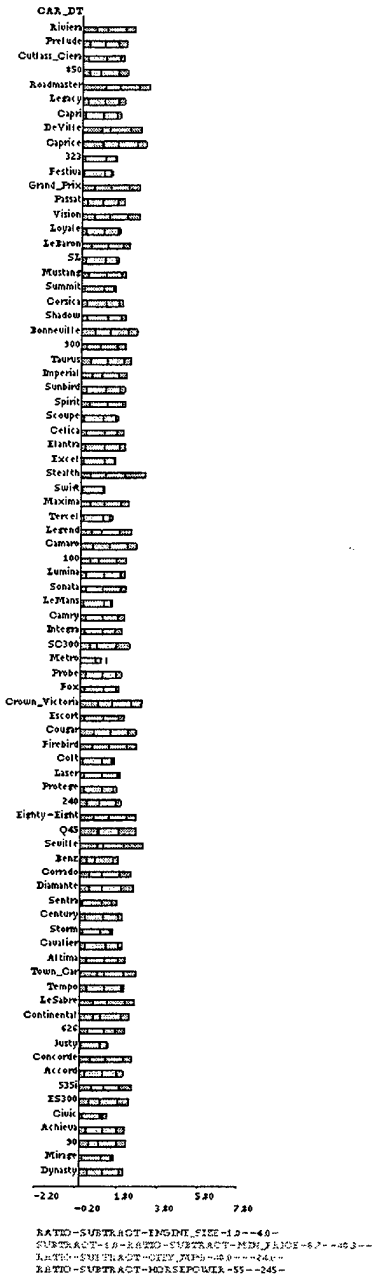
A better way to solve the task is to weigh each car based on all four attributes, and rank the cars based on the sum score of these weights (Task C-11). We will assume that all four categories are as important to Bob, so the weight is even for each category. To calculate the rank for each car, we score it from 0 to 1 for each attribute. The score for a particular attribute value is computed by determining where it falls between the minimum and maximum values for that attribute. 1 indicates the best value in the current data set and 0 indicates the worst value. Note that for the *price* attribute we need to perform an additional difference operation because the goodness of this attribute is inverse to its value size. The rank for a particular car is the summation of each individual attribute score. Since there are 4 attributes, the maximum score is 4.0.

```
Compute ( sum ,
    Compute ( Difference , 1.0,
        Compute ( ratio,   Compute (  Difference ,
                                      Lookup ( {all-cars}, price ),
                                      min-price )
                           Compute ( Difference , max-price,  min-price ) )
            )
    Compute ( ratio,   Compute (  Difference ,
                                  Lookup ( {all-cars}, mpg ),
                                  min-mpg ),
                       Compute (  Difference , max- mpg,  min-mpg ) )
    Compute ( ratio,   Compute (  Difference ,
                                  Lookup ( {all-cars}, engine-capacity ),
                                  min-engine-capacity )
                       Compute (  Difference ,
                                  max-engine-capacity, min-engine-capacity )
                             )
    Compute ( ratio,   Compute (  Difference ,
                                  Lookup ( {all-cars}, horsepower ),
                                  min-horsepower )
                       Compute (  Difference ,
                                  max-horsepower,  min-horsepower ) )
    )
```

Task C-11: Car purchasing task B
(Note that in this task specification we assume that the min and max values for each data attribute have been pre-calculated)



(a) A normalized score is computed for each of the car picking attributes and the sum scores are shown on the *x-axis* of the bar chart.



(b) A normalized score is computed for each of the car picking attributes and each score is represented by the *x-length* of a differently colored bar. The red bar represents the *engine-size* score, the green bar represents the *min-price* score, the blue bar represents the *mpg* score, and the purple bar represents the *horsepower* score. The sums can then be deduced by looking at the combined length of the stacked bars.

Figure C-22: Designs for car purchasing task B (Task C-11). Design (b) is more effective than design (a) because in design (b) it is easy to lookup the score sums and in addition, the individual scores are also given so that we may examine each criteria separately.

Because the task is computation intensive (i.e. many compute operators and task embeddings), the most effective design generated is a data computation design (Figure C-22a). In Figure C-22a, all the computations are pre-computed by the system and only the sum scores are shown for each car. While this method is much more effective than car purchasing task A (Task C-10), it requires that the user know what weightings to give to the different attributes. In addition, because the attributes are all rolled up into a single score, it is difficult to determine for a particular car which attributes contributed most to its final score. Figure C-22b shows an alternative hybrid design that is ranked second for this task. In this design, the score computation for each car attribute category is pre-computed, but the *addition* task for the four individual scores is achieved through a mapping transform. Each differently colored bar in Figure C-22b represents the score of a different car attribute, and the bars are stacked to facilitate the four score summations. This design solves the single score problem in Figure C-22a, however, the problem of identifying appropriate task weightings remain. In addition, both designs also suffer from having to display too many data elements, forcing users to navigate in the display to get to all the elements.

Note that a pure mapping design is much worse compared to the data computation designs in Figure C-22 because there are many embedded computations here and it is very difficult to graphically integrate all the information in a way that is consistent with the task constraints. In addition the added load from having to do all the intermediate tasks perceptually (i.e. embedded tasks) significantly increases the semantic distance of the design.

An issue related to task specificity is that of premature commitment. As we previously discussed, the more fully a task can be specified, the lower the articulatory load for the data computation design. However, the more that we need to commit at the outset of the task, the less flexibility we have later in the data analysis process. The importance of premature commitment depends on the task, the domain, and the user. We will show in the next section that our task specification language allows for less premature commitment by using the ('?') wild card character. This character indicates that the task parameter is unknown at the time of task specification. Building the task specifications themselves as well as deciding when it is appropriate to use wild card characters ('?'), however, is beyond the scope of this thesis.

Another way to perform the task is through a series of *finds*. We begin by finding a set of cars that have high mpg, and from there, we narrow down the set to those that have relatively low price and so on as in car purchasing task C (Task C-12), *case-1*. The problem with embedding tasks in this way is that it suggests an ordering to the tasks while in reality there is none. Thus in car purchasing task C (Task C-12), *case-2,* we use the *and* operator to group the *find* tasks together without embedding them.

Case 1:

```
Lookup
    (Find (  >  ,
           Lookup
                ( Find (  >  ,
                          Lookup
                           ( Find (  <  ,
                                     Lookup ( Find (  <  ,
                                                      Lookup (  all,  mpg ),  ?),
                                              price ),
                                     ? )
                            engine-capacity ),
                           ?),
                 max-speed ),
            ?)
    name )
```

Case 2:

```
Lookup (    And (   Find (  <  , Lookup (  {all-cars},  mpg ),  ?),
                    Find (  <  , Lookup (  {all-cars},  price ),  ? )
                    Find (  >  , Lookup (  {all-cars},  engine-capacity ),  ?)
                    Find (  >  , Lookup (  {all-cars},  max-speed ),  ?)
                    )
        )
```

Task C-12: Car purchasing task C

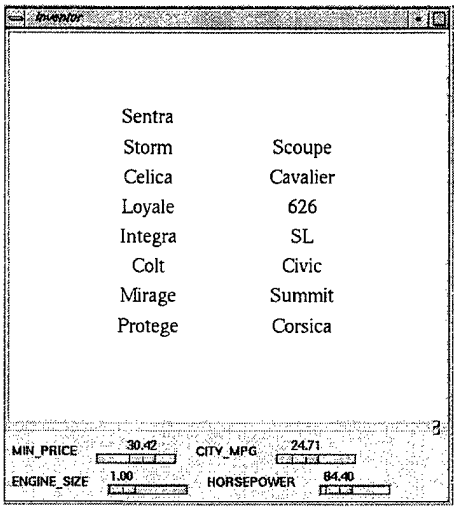


Figure C-23: Data computation design for car purchasing task 3
In this design users get the flexibility to enter in filtering thresholds for each of the four car picking attributes through *slider* input devices. The system then pre-computes all cars that fulfill those threshold conditions and then only displays those cars. This produces a much cleaner and effective design compared to the previous designs in Figure C-22. In addition, in this interface the user may weigh each of the four car picking attributes differently (e.g. place more importance on *horsepower* and less on *price*) by setting more or less stringent thresholds. In Figure C-22, each of the four car picking attributes are weighed equally and users are not given the ability to alter this weighting.

The best ranked design for car purchasing task C (Task C-12) is a data computation design (Figure C-23). In Figure C-23, four sliders are provided so that users may adjust the four car purchasing attribute thresholds. Only cars that fulfill the slider thresholds are shown as text in the display. Because of this data filtering, the number of cars shown at any one time is usually small. Even in the case where the original data set is large, a user can alter the threshold values so that most of the car concepts are filtered out except for the best choices. For this reason, the data computation design is ranked at the top even though car purchasing task C (Task C-12) has low task specificity. The data computation design has much less perceptual clutter compared to the mapping designs (Figure C-24 and Figure C-25) and it requires much less display space, thereby making it unnecessary to navigate through the visualization. In the mapping designs many more data concepts must be shown, and as a result users may need to scroll or zoom in and out to get to all the elements.

The top mapping design is shown in Figure C-24. In Figure C-24, the left bar chart encodes *engine-size* on the *x-axis* and *car-price* as *bar saturation*. The right chart encodes *miles-per-gallon* in the city on the *x-axis* and *horsepower* as *bar saturation*. The *car-names* are mapped to the *y-axis* of both charts. To perform car purchasing task C (Task C-12), we must identify bars in the first chart that are long (large *engine-size*) and unsaturated (low *car-price*). They must be paired with bars in the second chart that are long (high *miles per gallon*) and saturated (high *horsepower*). Compared to Figure C-23 many more graphical elements need to be shown here, thus users will most likely need to navigate within the visualization to get to all the elements. In addition, *saturation* does not allow value comparisons to be performed accurately. For these reasons, the data computation design in Figure C-23 is ranked higher than the pure mapping design in Figure C-24.

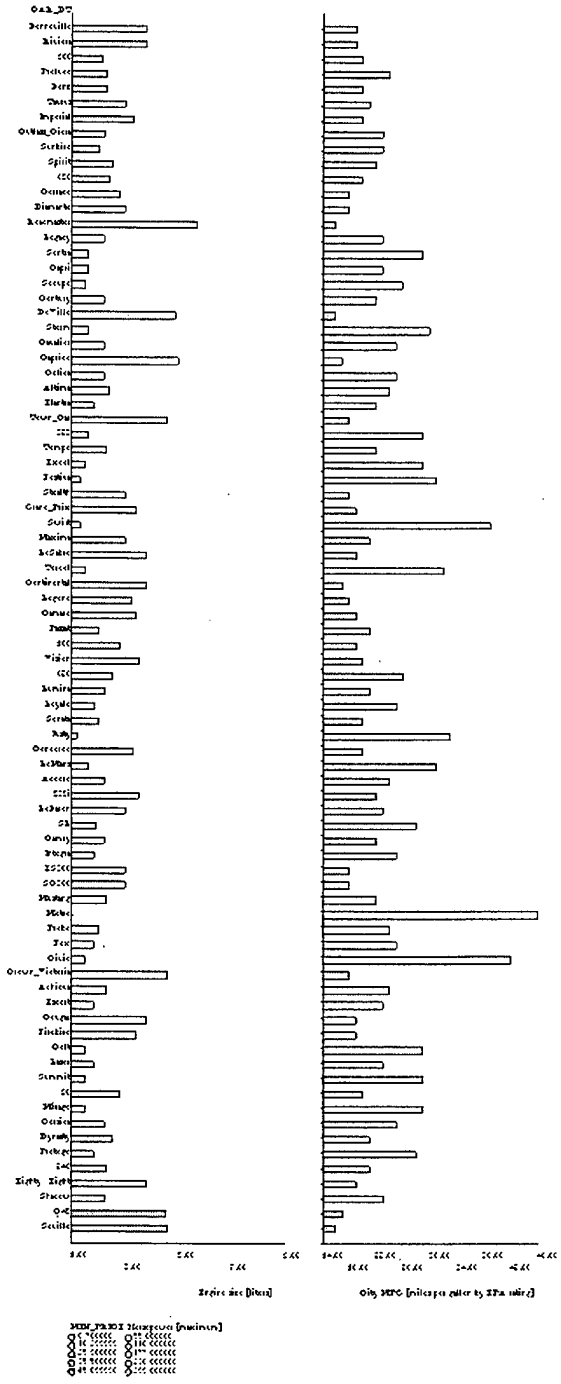


Figure C-24: Mapping design-1 for car purchasing task 3
Mapping design that shows the original four stock picking attributes to the user. *Engine-size* is mapped to the *x-axis* of the left chart and *min-price* is mapped to the *saturation*; *city-mpg* is mapped to the *x-axis* of the right chart and *horsepower* is mapped to *saturation*. Desirable cars are those with long (large *engine-size*), unsaturated (low *min-price*) bars in the left chart and short (low *city-mpg*), saturated (high *horsepower*) bars in the right chart. This design is significantly more complex in terms of number of elements and difficulty of interpretation compared to Figure C-23. In addition the *saturation* encodings does not allow for accurate value lookups or comparisons

An alternative design is to map each attribute to a set of aligned bars (Figure C-25). Figure C-25 is deemed less effective compared to Figure C-24 because it requires users to scan across a wide spatial distance to get to each aligned value. In addition, an even greater amount of navigation is probably needed here due to the larger visualization size. However, by using *position* to encode all of the attributes, we can perform value comparisons much more accurately than in Figure C-24.

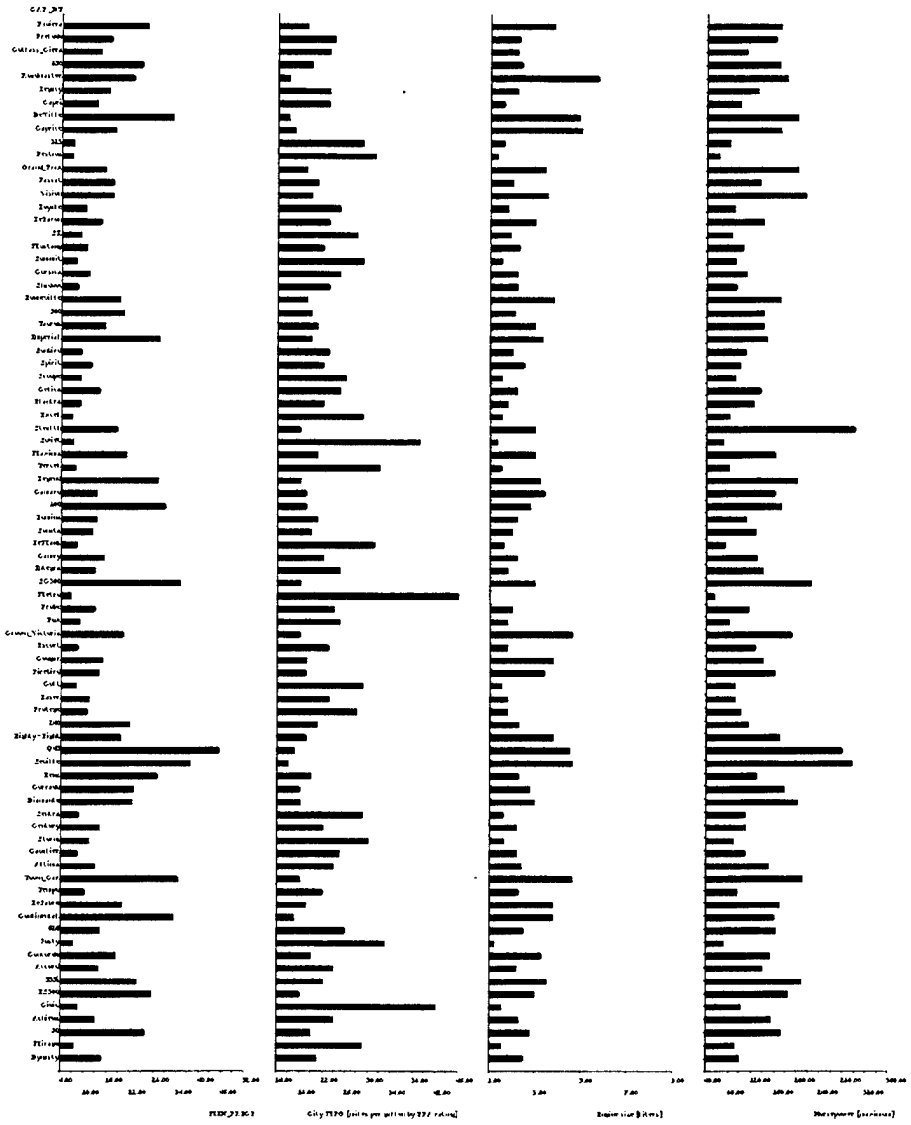


Figure C-25: Mapping design-2 for car purchasing task 3
Mapping design that shows the original four stock picking attributes to the user. *min-price* is mapped to the *x-axis* of chart 1 (left-most chart), *city-mpg* is mapped to the *x-axis* of chart 2, *engine-size* is mapped to the *x-axis* of chart 3, and *horsepower* is mapped to the *x-axis* of chart 4 (right-most chart). Desirable cars are those with short bars in chart 1 (low *min-price*), short bars in chart 2 (low *city-mpg*), long bars in chart 3 (large *engine-size*), and long bars in chart 4 (high *horsepower*). This design is more accurate than Figure C-24 because all values are encoded on the *x-axis* (i.e. no *saturation* values are used). However this design is also less integrated and requires significantly more eye movement, display space, and display navigation.

A better mapping alternative is to integrate the attributes together within a single graphical object so that users need not read graphical properties across different objects across multiple spaces. In this case however, none of the graphical objects available to the designer is capable of expressing five attributes (*name, mpg, price, engine-capacity, max-speed*) in an integrated manner.

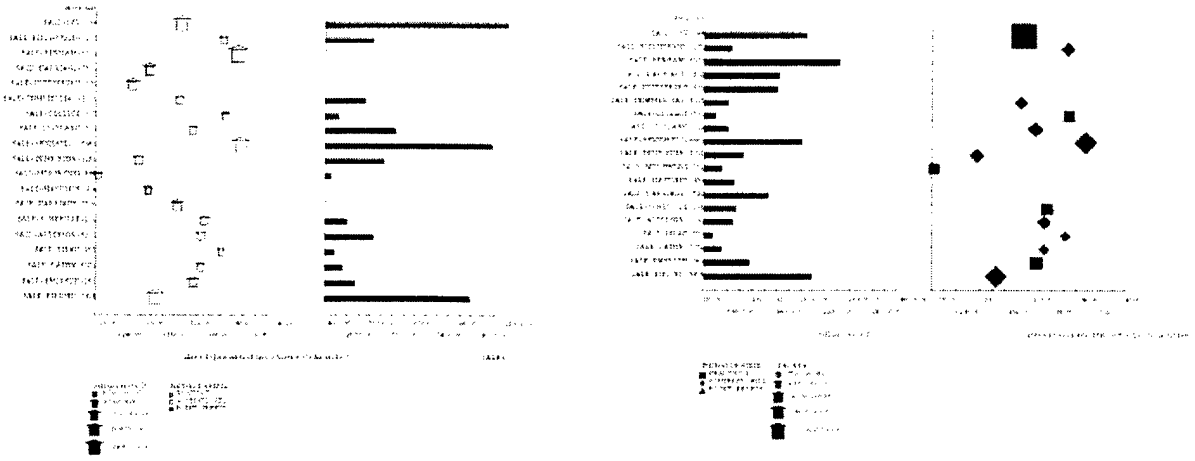
This example illustrates how a design that most suits the user's needs can be derived through iterative refinement between the user and the automatic design system. An automatic design system supports this process, by enabling users to rapidly test out different design ideas, and by presenting design alternatives that can serve as starting points for further refinements. As we had discussed previously, the ultimate goal is not for users to learn and specify the task constructs shown here, but rather to attach the automatic designer to a higher level domain specific system, that can generate these specifications based on verbal descriptions by the user or based on the design alternatives that are favored by the user.

Appendix D

Appendix to Implementation (Chapter V)

D-1 Structural & Content Matching

Structural matching is based on the graphical properties used, the number of graphical objects, and the number of graphical regions in the design. For a structural match to occur, the number of graphical objects and the distribution of those objects across the different regions must be identical for the two visualization designs. For example Figure D-1a and Figure D-1b match structurally because both visualizations have two *chart* regions and a set of *mark* graphical objects in one region and *horizontal-bar* graphical objects in the other. Note that the order of the two regions is irrelevant in the structural match.



(a) (b)

Figure D-1: An example pair of visualizations that match based on both structure and content. Structurally both designs have two charts, one of which is a bar chart and the other a scatterplot. In terms of content, both designs contain the same data attributes (*object-name, selling_price, neighborhood, owner_salary*) and graphical property classes (*2 positionals,* and *2 retinals*).

Content matching, on the other hand, is based on the data contained within a visualization and how that data is mapped to graphical properties. Two designs match in content if the *data attributes* and *graphical property classes* contained within them are identical. There are three graphical property classes in AVID (based on Bertin's [*Bertin, 1983*] graphical property categorization): *positional, retinal*, and *labels*. *Positional* properties include *x-position, y-position,* and *z-position*, *retinal* properties include *size, hue, saturation, shape, thickness,* etc., and *label* properties include the use of *text labelling*.

Figure D-1a and Figure D-1b are also content matches because both visualizations have identical data content (*object-name, selling_price, neighborhood, owner_salary*) and identical graphical property class content. Figure D-1a uses *y-position, size, hue* and *x-position* respectively to encode its data attributes which translates to a graphical property class content of [*positional, retinal, retinal, positional*]. Figure D-1b uses *y-position, x-position, shape* and *size* to encode its data attributes which translates to a graphical property class content of [*positional, positional, retinal, retinal*]. Note that the graphical property class content of Figure D-1a and Figure D-1b match even though the actual graphical properties used may be different. Figure D-1b for example uses *shape* to encode *neighborhood* while Figure D-1a uses *hue* to encode the same data attribute. A match still occurs because both *hue* and *shape* belong to the *retinal* property class. Also note that the order in which data attributes are mapped to graphical properties does not affect the content match result. For example in Figure D-1a, a *retinal* property (*size*) is used to encode *selling_price* and a *positional* property (*x-position*) is used to encode *owner_salary*. On the other hand in Figure D-1b a *positional* property (*x-position*) is used to encode *selling_price* and a *retinal* property (*size*) is used to encode *owner_salary*. Even though the mappings are permuted in these two cases the graphical property classes used and the data attribute content within the two visualization designs are identical, thus Figure D-1a and Figure D-1b are considered content matches.

D-2 Translating a Functional Design from AVID's Design Component into a Complete Specification

The translation algorithm for AVID's functional design follows the *instantiation augmentation process* for interactive techniques described in chapter III. Initially we collect all functional operators in the *functional-operator-list* of a node and connect them based on the embedding structure of their related tasks. For example consider the design in *node-3*, which is a pure data transform design with five data transform operators corresponding to each of the input tasks. These operators are connected from innermost task to outermost task. Thus the *GetAttributeValue selling_price* operator for the innermost *lookup* task is connected to the *Threshold* operator of its parent *find* task. The *Threshold* operator is in turn connected to the *GetAttributeValue* operators for the *lookup-date_on_market* and *lookup-date_sold* tasks. Finally we end the functional design with the *BinaryCompute* operator related to the outermost *compute* task. This initial design structure is shown in Figure D-2. All functional operators are represented as rectangular boxes.

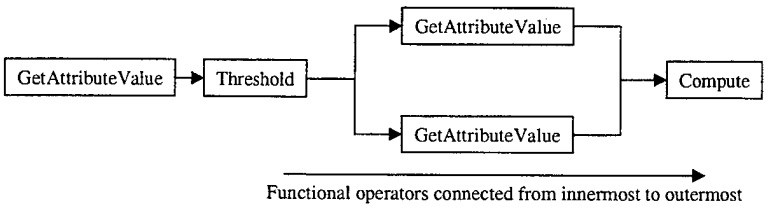


Figure D-2: Connecting all visualization functions within a node state from innermost task to outermost

Once we have constructed the general functional structure, we perform *task class* related modifications, as is shown in Figure D-3. Newly added function rectangles are highlighted to indicate changes made to the design specification at the current step. Primarily, a *find* task can either show its results through object filtering or through mapping its results to a graphical property. Depending on which of these alternatives is relevant to the current design, we either filter objects within the appropriate region with an *Add_object* and a *Delete_object* operator (*alternative 1* in Figure D-3) or we store the *Threshold* results within the visually mapped *find* attribute with an *Assign* operator (*alternative 2* in Figure D-3)[1]. In Figure D-3, an *Assign* operator is also connected to the *BinaryCompute* operator. This *Assign* operator stores processed time duration values within the *compute* task's related *duration_on_market* attribute that is subsequently mapped to graphics. Note that both *find* task design alternatives are shown in Figure D-3 for pedagogical purposes. Any single design, however, would only contain structures for **one** of these alternatives

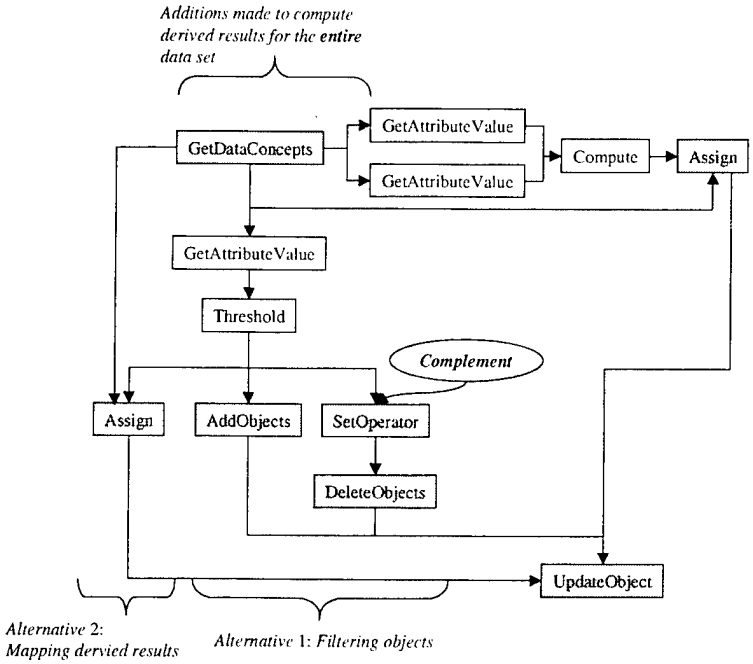


Figure D-3: Making task class modifications depending on whether we want to show the *find* task results either through object filtering or by mapping its results to a graphical property

Depending on how the *find* task results are shown, it may be necessary to show the computed time duration values only for the objects that fulfill the *find* task (*design-1*) or we may have to show the values for all house

[1] In general all visualization functions that provide values to mapped data attributes must be linked to an *Assign* operator so that newly computed values may be associated with their corresponding data attribute structure.

concepts (*design-2*). For design simplicity, we always calculate derived attributes for the entire set of related data concepts. To achieve this we disconnect all embedded object task operators from their parent task and connect them to a *GetDataConcepts* function instead. In Figure D-2, for example, the *Threshold* function results (for the *find* task) are piped into two date *GetAttributeValue* operators that limit the duration *compute* to only the threshold objects. To generate duration values for the entire house data set we disconnect the out-links from the *threshold* operator, and associate the two date *GetAttributeValue* operators to a *GetDataConcepts* operator. The *GetDataConcepts* operator accepts a set of data classes and extracts all data concepts belonging to those classes from the entire house data set. In this example derived values are calculated for all data concepts belonging to the house class[2].

Finally we must update the visualization design so that any newly computed results or change in values are reflected in the graphics. This is achieved by adding an *UpdateObject* operator at the end of the functional specification. We connect to the *UpdateObject* operator last to ensure that all computations and assignments are made before the object update occurs. The procedure thus far corresponds to step 1 of the *instantiation augmentation process* described in chapter III. In this step we define the object selection and transformation functions in the design[3].

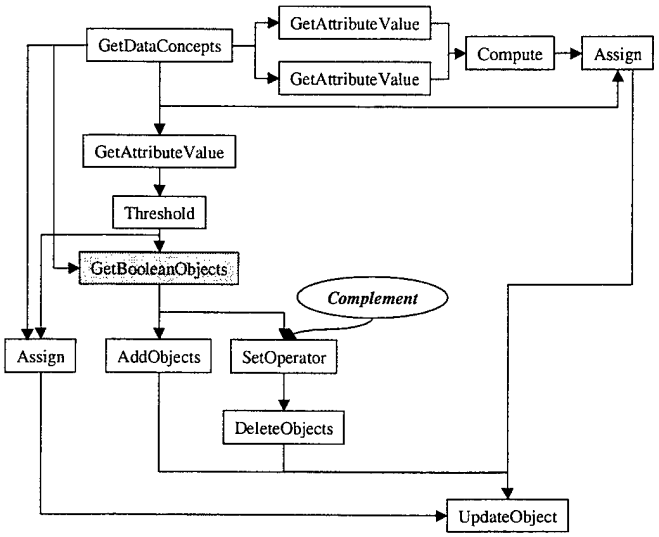


Figure D-4: Adding in any necessary translation functions

---

[2] Note that the *GetDataConcepts* operator also produces the input data concepts for the two *Assign* operators that are used to store computed functional results within the *duration_on_market* attribute.

[3] In Figure D-2 and Figure D-3 we include *GetAttributeValue* functions that are technically translation functions but since they are related to the *lookup* tasks we found it useful to add them to the design at this initial step.

In step 2 of the *instantiation augmentation process* we add in translation functions to ensure that the inputs to a visualization function match the output of its source function. In this case only one translation function is added, namely the *GetBooleanValues* function for converting boolean values from the *Threshold* function into an object set so that it can subsequently be processed by the *Add_object* and *Delete_object* mapping functions (Figure D-4).

Next, we determine the input arguments to the visualization operators based on the tasks and task arguments that they are associated with. This corresponds to steps 3 and 4 of the *instantiation augmentation process*. These input values are represented in Figure D-5 with oval boxes and ***bold italicized*** text. Most of these values can be directly extracted from the task arguments (e.g. *selling_price, 100k, date_on_market,* and *date_sold*). The two derived attributes *duration_on_market* and *houses_with_selling_price_less_than_100k* can have their names specified in the task description or automatically generated based on related task operators and arguments. The region name(s) for the *Add-object* and *Delete-object* functions is derived from object constraint information stored within the node state during the search procedure. In general all regions containing attributes from an object task (e.g. *find* or *AND*) are constrained by the results of that object task and thus become inputs to its corresponding *Add-object* and *Delete-object* functions. In *design-1*, for example, the *interval-bar region* contains the *duration_on_market* derived attribute, which is a parent attribute to the *find* task. Thus the *interval-bar region* gets assigned as the input region to the *Add-object* and *Delete-object* functions in Figure D-5. Finally the entire visualization design (*design-1*) is updated using the *UpdateObject* function.

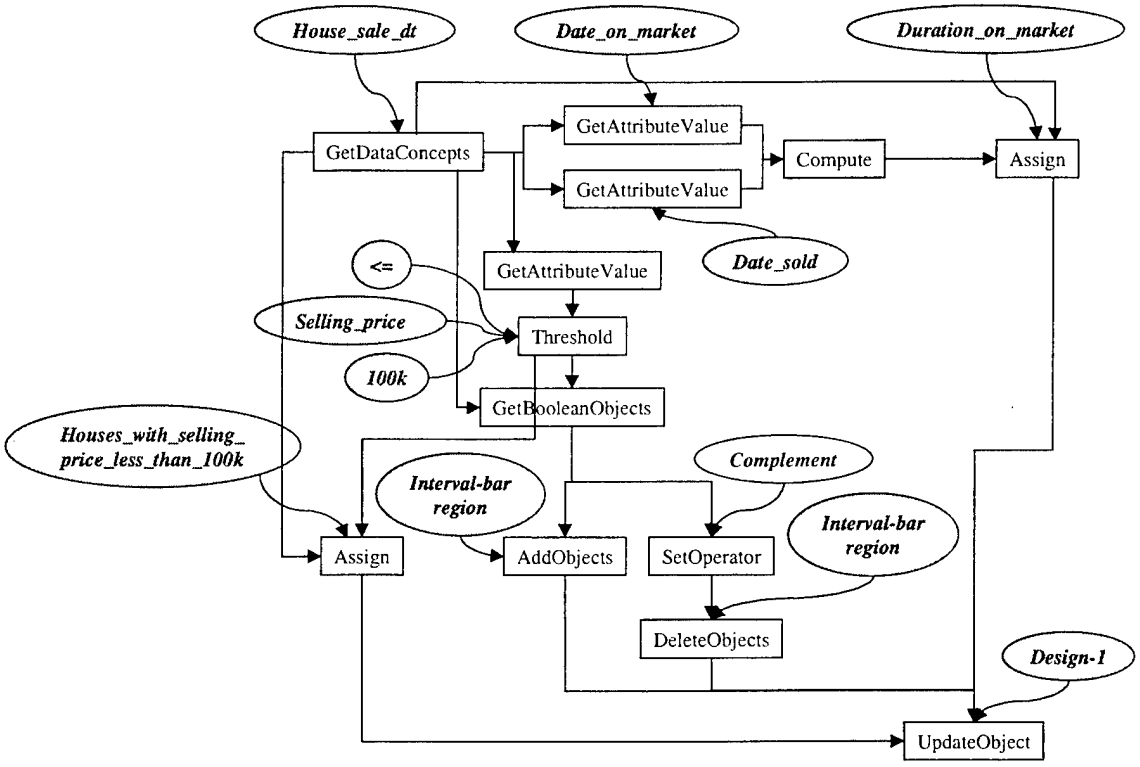


Figure D-5: Adding in input arguments to the visualization functions

In step 5 of the *instantiation augmentation process* we add in all the input devices stored within the node state and initialize them. The house purchasing task in chapter V, Figure V-2, does not require the use of any input devices, thus all designs (*design-1,2,3,4,5,6*) generated have none. For illustration purposes however, suppose that the *find* threshold value is tied to an input device instead of a pre-specified constant of *100k*. In this case, a *slider* input device would be associated with the *find* task and stored within the *input-device-list* of the node state. We therefore add the device to our functional specification and augment it with any initialization functions necessary.

For each input device class, AVID contains knowledge on the set of device attributes that must be initialized (this is based on the input device description in chapter IV). For a *slider*, we need to initialize three attributes: its *label name*, as well as its *min* and *max* values. Figure D-6 shows the initialization specification for the *slider* input device. To the right are the three device attributes that must be initialized. To the left we extract relevant values from the task argument(s) related to the input device. In this example the related task argument is associated with the *selling_price* attribute and the house object class. As shown in Figure D-6 (highlighted ovals), these values are piped into the input device initialization specification. The rest of the functions and inputs within the initialization structure are stored as design knowledge within AVID. Similar to the *slider*, other devices have initialization structures associated with them as well, including entry point rules for connecting task argument information.

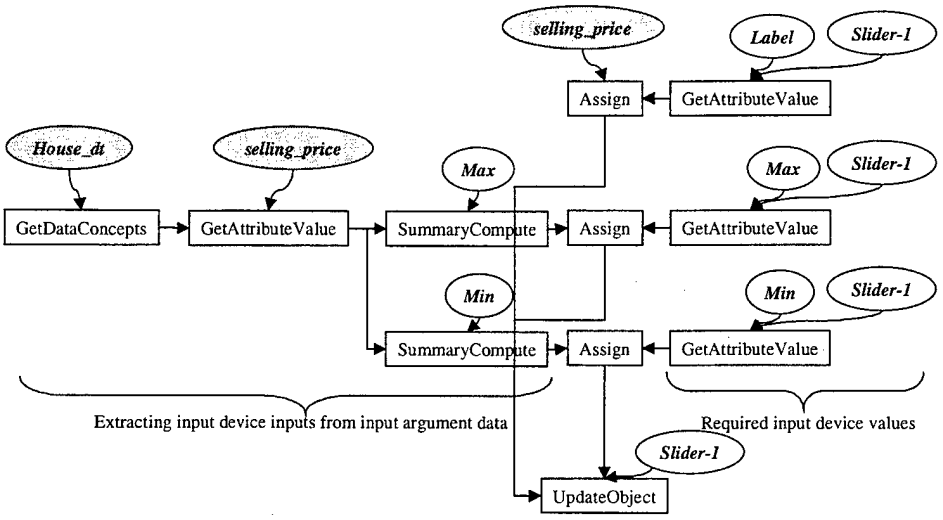


Figure D-6: Initialization functions for the slider input device

D-3 Visualization Realizer Component

AVID's realizer is divided into two components: 1) the graphical object realizer and 2) the functional realizer. The graphical object realizer accepts visual structure design specifications and converts them into graphical element renderings. The functional realizer accepts functional design specifications and converts them into visualization techniques (e.g. dynamic query sliders, painting).

D-3.1 Graphical Object Realizer

The graphical object realizer is implemented in C++, using SGI's Inventor toolkit, which provides a framework for organizing and rendering the graphical objects within the visual structure specification. Initially, C++ functions are in place to interpret the input design and convert it into one or more Inventor nodes. These Inventor nodes can then be manipulated or rendered onto the display using a set of Inventor functions.

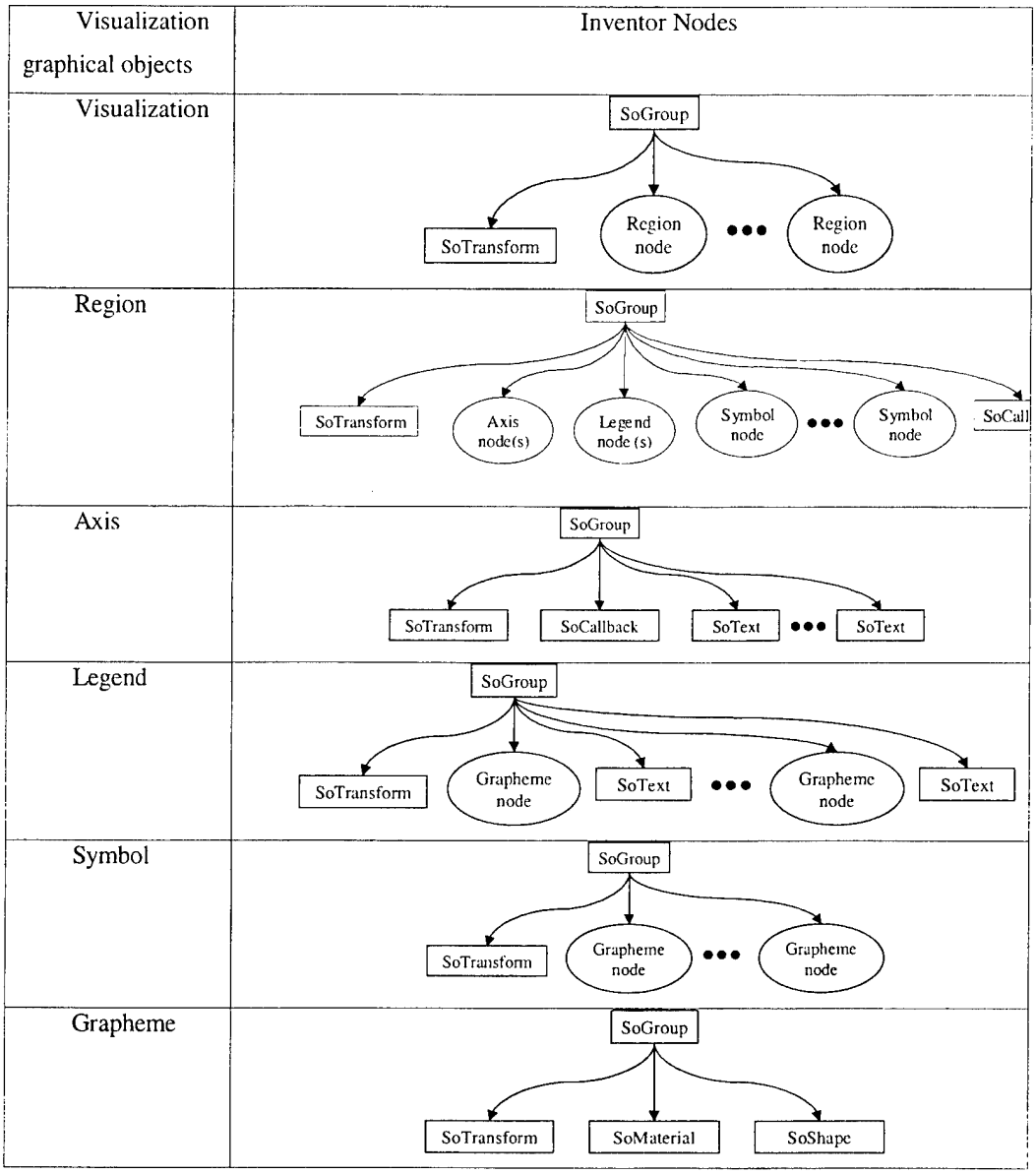


Figure D-7: Visualization graphical objects and their corresponding Inventor nodes

Figure D-7 shows all the graphical object classes in our AVID system and their corresponding Inventor nodes or sub-trees. Details on these object classes can be found in chapter III-1.1.2. Primitive Inventor nodes are

represented in Figure D-7 as square boxes and their names are preceded with *"So"*. Oval boxes represent AVID graphical object classes.

All graphical objects in the visual structure specification are converted into at least a *SoGroup* and a *SoTransform* Inventor node. The *SoGroup* node is used to collect all other nodes associated with the graphical object together under a single root node. This makes it easier to organize the objects within a visualization as well as to access and manipulate their appearance. Every *SoGroup* node has a *SoTransform* node as its first child. This *SoTransform* node allows the realizer to translate, rotate, or scale graphical objects. A change in the transformation parameters of a *SoTransform* node not only changes the appearance of the current graphical object but also the appearance for all its children. For example, increasing the scale of a *SoTransform* node for a *region* object increases the region bounds as well as the size of the *axes*, *legend*, *symbol*, and *grapheme* objects within it. Changing the scale of a *SoTransform node* for a *symbol* object will affect the size of the *grapheme* objects it contains but not have any effect on the *region,* or *visualization* objects which contain the changed *symbol*.

In addition to the *SoGroup* and *SoTransform* nodes, graphical objects may also contain other graphical objects. In fact the graphical object classes within AVID have a hierarchical relationship with *visualization* objects at the top of the hierarchy followed by *region* objects, *symbol* objects and finally *grapheme* objects. Depending on the region class (i.e. *chart, map, table* or *grid*) and the *grapheme* properties used, regions may also contain positional *axis* objects and *legend* objects that capture how *grapheme* property values can be converted back into data values. *Grapheme* objects form the leaves of the hierarchy, and as such they only contain primitive Inventor nodes, unlike the other graphical objects. *Grapheme* objects have three Inventor nodes: a *SoTransform* node that determines the position, orientation and size of the object, a *SoMaterial* node that determines the color of the objects, and a *SoShape* node that determines the appearance of the *grapheme* object. The appearance of a *grapheme* object is based on its class (*mark, bar, text, interval-bar* or *line*) and its *shape* graphical property.

Certain graphical objects also contain Inventor *SoCallback* nodes. *SoCallback* nodes are free-form nodes that allow specialized drawing functions to be executed. In the *region* object, for example, this node is used to call functions for texturing maps, drawing table columns, and other region rendering operations. Similarly the *SoCallback* node for the *axis* object is used to draw the axis line, and the axis tick marks.

When we connect the Inventor nodes for the different graphical objects together, we get a hierarchical *scene graph* of Inventor nodes, as in Figure D-8. This scene graph is rooted at the *SoGroup* node of a *visualization* object. Every *visualization* object in a design gets converted into a separate hierarchy of objects. Once the realizer converts all objects within the visual structure specification into a *scene graph*, Inventor functions are available to render the scene onto a display screen and ensure that all object transforms are applied in proper order.

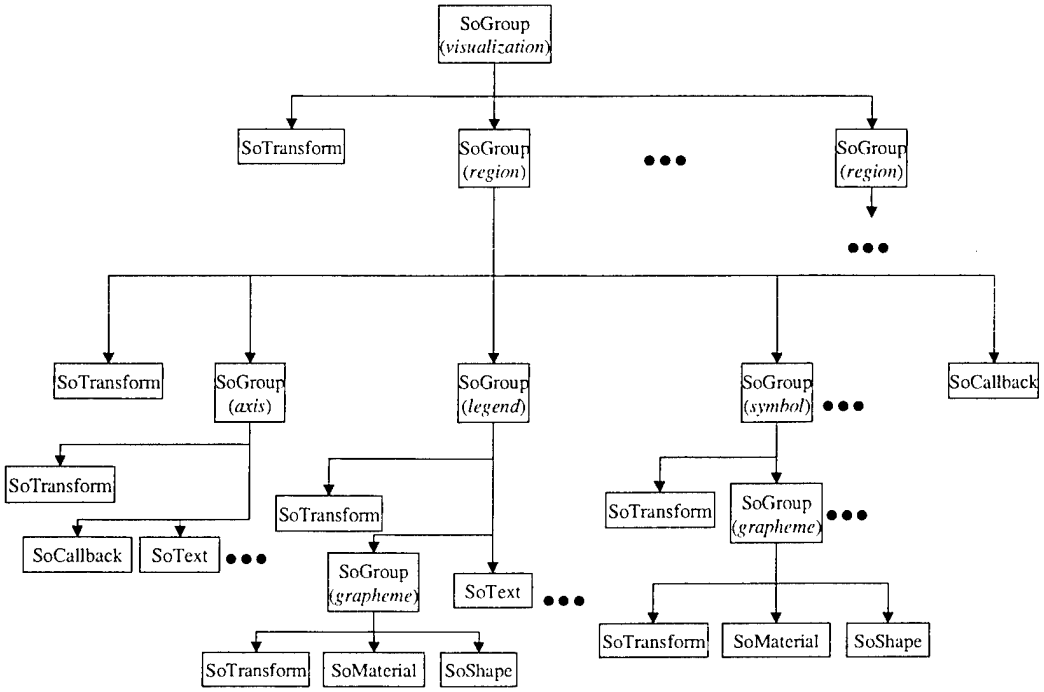


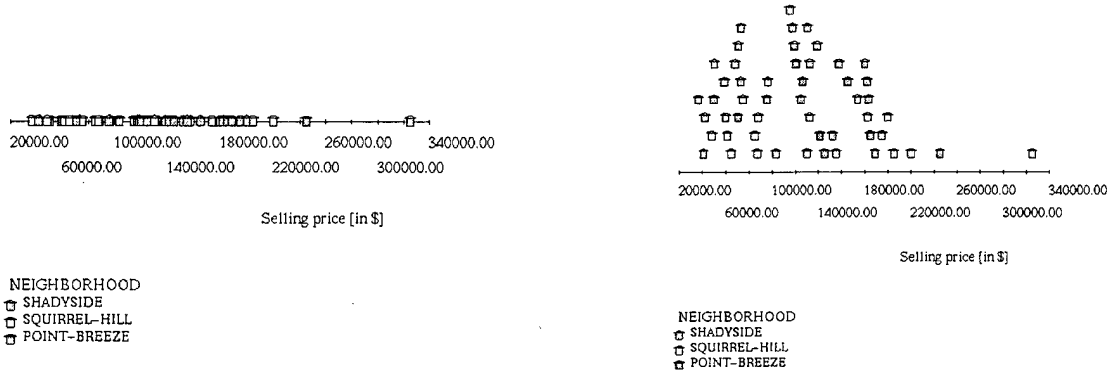
Figure D-8: Inventor scene graph of visual structure design

Another important function of the graphical object realizer is in making layout decisions. Note that the visual structure specification provided to the realizer does not contain details on exact object positions. Instead, the specification contains mapping information such as which data attribute should be mapped to which graphical objects and properties, object information such as what objects should be in the visualization, and roughly how objects should be laid out in relation to one another, as well as containment relationships. It is up to the realizer to set the actual pixel or color ranges used for mapping, the actual object sizes and positions in pixels as well as default object property values. To avoid certain occlusion problems among the graphical objects and to reduce perceptual clutter, the realizer has a set of algorithms for arranging graphical objects. For example, white-space is added between *region* objects, and *regions* are never placed on top of one another. *Grapheme* objects may also be subject to special layout algorithms. This occurs in *single-axis charts* or when a chart maps non-unique, non-continuous data attributes to positional axes.

D-3.1.1 Single axis layout

For single axis charts, occlusion may be significant because the objects are only distributed across a single positional dimension. For example Figure D-9a shows a single axis visualization with *selling_price* mapped onto the *x-axis*. Because of the great concentration of objects around particular ranges of values there is significant occlusion in those areas which in turn reduces readability of the visual display. The AVID realizer reduces occlusion in this case by utilizing the other, unmapped positional dimension (i.e. the *y-positional* axis) to spread out the high object concentration areas. This is achieved by adding an offset value to the unmapped positional dimension of all

occluding objects. We repeat this positional offset step until there is no longer any occlusion. This pushes occluded objects up as far as necessary, as is shown in Figure D-9b which has the same data and design as Figure D-9a, but with the *single axis* layout algorithm applied.

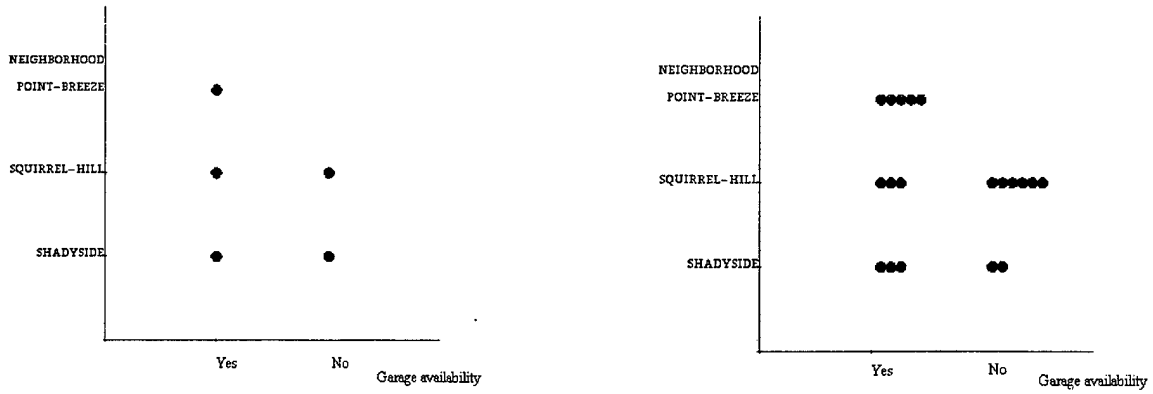


(a) without layout **(b) with layout**

Figure D-9: Single axis visualization design with and without realizer layout algorithm

D-3.1.2 Non-unique positionals

Significant occlusion may also occur among *grapheme objects* when a chart only contains non-unique, non-continuous positionals. For example, in Figure D-10a there is significant occlusion because two non-unique data attributes, *neighborhood* and *garage_availability* are mapped to the *y-axis* and *x-axis* respectively. As a result all houses that fall within the same *neighborhood-garage_availability* category will have exactly coincidental positions. In such cases the graphical object realizer offsets the positions of objects within each category so that they do not occlude one another as in Figure D-10b which contains the same data and design as Figure D-10a but with significantly less occlusion.



(a) without layout **(b) with layout**

Figure D-10: Non-unique positional visualization design with and without realizer layout algorithm

D-3.2 Functional Realizer

The functional realizer takes a functional design from the automatic designer as input and produces active visualization techniques for manipulating data or graphical objects. AVID's functional realizer is implemented in C++ and Motif. Initially C++ functions are used to transform the objects within the functional specification into networks of data transform functional nodes. Each network is a *directed acyclic graph*, and may look like the specifications in Figure D-5 or Figure D-6. The Motif toolkit is used to integrate virtual input devices and handle events from physical devices such as keystrokes, mouse operations, etc. These Motif events serve as alerts and may cause one or more functional networks to be executed. When a functional network is executed, we traverse down the acyclic graph and execute the transform functions accordingly.

The functional network structures in the specification are based on the framework described in chapters II and III of this thesis. There are four types of objects in this network:

D-3.2.1 Primitive visualization functions

A *visualization function* (VF) structure describes the primitive building blocks of a visualization technique. Figure D-11 shows an example visualization function structure in our functional language.

```
(DEFSCHEMA visualization-function-741
    (class BINARY_COMPUTE)
    (defaults
      (0 0 0 0 0 VALUE IV_STRING SUBTRACT)
      (0 0 0 0 1 VALUE IV_STRING ONE-TO-ONE)
      )
    (to con-743    )
    (composite cvi-748)
    )
```

Figure D-11: An example description of a visualization function

The *class* field captures the primary processing operations of the visualization function. As was described in chapter II, visualization functions may belong to the object selection class (*object-definition* or *enumeration*) or the transformation class (*data, mapping, graphical,* or *rendering* transforms). Tables III-1, III-2, III-3, and III-4 in chapter III summarize all the visualization function classes defined in our AVID realizer. Each function class definition contains the number and type of input arguments required, the number and type of output arguments generated, and a functional description (in C++ code) of how the input arguments are processed. When a visualization function is activated, its input arguments are processed using the functional description code defined within its function class.

A system designer may provide input arguments to a visualization function as default values. For example the visualization function in Figure D-11 has two default input value strings, *subtract* and *one-to-one*. Input arguments may also be generated by other visualization functions. This is achieved by attaching both the source and destination visualization functions together using a *connector* object. For example, in Figure D-11, *visualization-function-741* is connected to the connector object, *con-743*. Figure D-12 shows that this connector object routes the output of *visualization-function-741* to *visualization-function-742*.

D-3.2.2 Connectors

A *connector* routes output arguments from one visualization function as input arguments into another. Figure D-12 shows an example connector object. A *connector* object contains a source and a destination visualization function in its *visualization-function* slot. This information alone however, is insufficient to fully specify the connection. A visualization function commonly has multiple input and output arguments, thus to fully specify a connection, we must not only declare the source and destination functions, but also the specific output and input argument positions. This information is stored in the *connections* field. In Figure D-12, *con-743* links the first output argument of *visualization-function-741* to the second input position of *visualization-function-742*. The first number in the *connections* field represents the number of connections there are in the field. The subsequent numbers in the field are pairs of source argument and destination argument positions.

```
(DEFSCHEMA con-743
     (instance bvi-connection)
     (visualization-function  visualization-function-741
                              visualization-function-742)
     (connections 1 1 2 )
     )
```

Figure D-12: An example description of a connector

When the *connections* field of a connector object is left unspecified, the functional realizer will try to infer which argument(s) of the source function best matches any unspecified input arguments in the destination function based on the type and properties of those arguments.

D-3.2.3 Input devices

An example input device structure is shown in Figure D-13. The input device *class* describes the type of input device to use, which could be a physical device, e.g. *mouse, light-pen* or a virtual device *option-menu, scroll list* or *slider*. In addition to the input device class, we must also declare the *trigger visualization functions*. The *trigger visualization functions* are the functions that get executed whenever certain *trigger events* are sensed by the device. Trigger events commonly differ based on the input device class. For example the trigger event for a *scroll list* is a double click on a menu choice, the trigger event for an *option button* is a mouse release on one of the button choices and the trigger event for a *bounding-box* is a mouse release within the visualization window. The types of input

devices available on our AVID system and their corresponding trigger events are shown in Table III-5, chapter III. Finally, the input device structure also contains information on which visualization window the input device should be attached to. In Figure D-13, the input device *id-500* is attached to the visualization *design-338*.

```
(DEFSCHEMA id-500
    (instance inputDevice)
    (class BOUNDING_BOX)
    (trigger bvi-500)
    (operation-within design-338)
    (composite cvi-501)
    )
```

Figure D-13: An example description of an input device

For each input device class, AVID's realizer contains information on its trigger event, its appearance, its properties as well as a functional description of how the device's properties and state change with respect to user inputs.

D-3.2.4 Composite visualization functions

As is shown in Figure D-5 and Figure D-6, a set of visualization functions and input device objects may be combined together through a set of connector objects to form a functional network (which is a directed acyclic graph). Each functional network is called a *composite visualization function*. A visualization technique may consist of a single composite or multiple interacting composites. In the example in section D-2 there are two composite functions, one for providing the inputs to the slider input device and the other for describing how the visualization graphical objects are altered when the user enters different values through the slider. Figure D-14 shows an example composite visualization function (*composite-501*). This composite object contains all the source or origin visualization functions within the functional network. Source visualization functions are operators that only contain pre-defined input arguments, i.e. operators that do not accept inputs from other visualization functions. When a composite visualization function is activated, processing begins with these source functions.

```
(DEFSCHEMA composite-501
    (instance composite-function)
    (source-nodes visualization-function-734
                  visualization-function-726
                  visualization-function-751)
    )
```

Figure D-14: An example description of a composite visualization function

Based on the structural descriptions of these four object types within the input functional specification, the functional realizer builds one or more networks of connected *visualization functions* and *input device objects*.

Initially, all functional networks are activated once beginning with their source visualization functions. In addition to this initial activation, functional networks may also get reactivated as a result of trigger events from input devices. When a *visualization function* is activated its input arguments are transformed based on its function class. The newly generated outputs are piped into subsequent visualization functions that in turn get activated. Note that a visualization function can only be activated when all of its input functions (i.e. the functions that generate the inputs to the current function) have also been activated. This ensures that all of the input data are up to date before processing begins. If an error in the design specification results in un-updated or missing input arguments, then processing of the error visualization function, as well as its parent composite function is halted.

D-3.3 Summary & Scope

This section describes the realizer component of AVID, which is divided into two parts, the graphical object realizer and the functional realizer. The graphical object realizer deals with rendering the graphical components within a visualization design while the functional realizer deals with constructing and executing the selection and transform visualization functions associated with a design.

The selection and transformation functions captured within our functional realizer are merely a subset of the wide range of possible useful data analysis operations. In fact as we discussed in earlier chapters, it is never possible to guarantee complete coverage of the functional design space nor the visual design space. However, the designer and realizer components within AVID are modular in their implementation, and it is not difficult to add in new visualization function classes, input device classes, graphical object classes and region types.

D-4 Interactive Functions Editor

In the previous section we described AVID, our automatic visualization designer that can generate a set of ordered visualization designs (consisting of both visual and functional components) based on input task specifications. AVID makes design decisions about what graphical object and visualization function primitives to use and how they should be combined. AVID's designer component, however, does not utilize the entire visualization techniques design space that is captured by our framework in chapters II and III. In AVID's designer component we mainly focussed on heuristics for integrating data manipulation, data summarization, and mapping techniques. We chose to leave graphical and rendering heuristics for future work, as this area is both rich and complex, making it difficult for us to give a complete or reasonable treatment of it in this dissertation. In appendix F we discuss graphical and rendering techniques in greater detail and outline how they may be integrated into the automatic design process in future work.

To more widely test the flexibility of our functional framework and its usability as a prototyping and customization language we implemented an editor for manually constructing visualization techniques. This editor allows users to build functional networks using the primitives described in chapters II and III. It is built upon two graph packages developed at AT&T-Bell Labs: *GraphViz* and *JavaApp* [*Ellson*]. *GraphViz* is used to improve the

readability of a graph by organizing it so that there is little or no occlusion among its nodes and links. *Java App* allows visualization techniques to be constructed by adding visualization function nodes (indicated by rectangles), default value nodes (indicated by ovals), and connector objects; changing visualization function properties (using property sheets); deleting or moving existing nodes and other graph editing operations. We made some modifications to the *Java App* interface to better suit our purposes, such as allowing group selection, group moves, group deletes, as well as automatic resizing of the graph canvas depending on the current graph size. Figure D-15 shows a screen-shot of our functional specification editor. The top window allows us to load and save functional specification files, remove or add *cvi-components* to the current design, or layout chosen function networks. The bottom window shows the network diagram for a particular *cvi-component* (*cvi-4*) and a property sheet of one of the *visualization functions* within it. Once a functional specification is completed, it is transformed into the functional specification language that we described in section D-3.2. This specification may be saved as a file and/or piped into AVID's realizer component that instantiates the designs and combines them within a visualization interface.

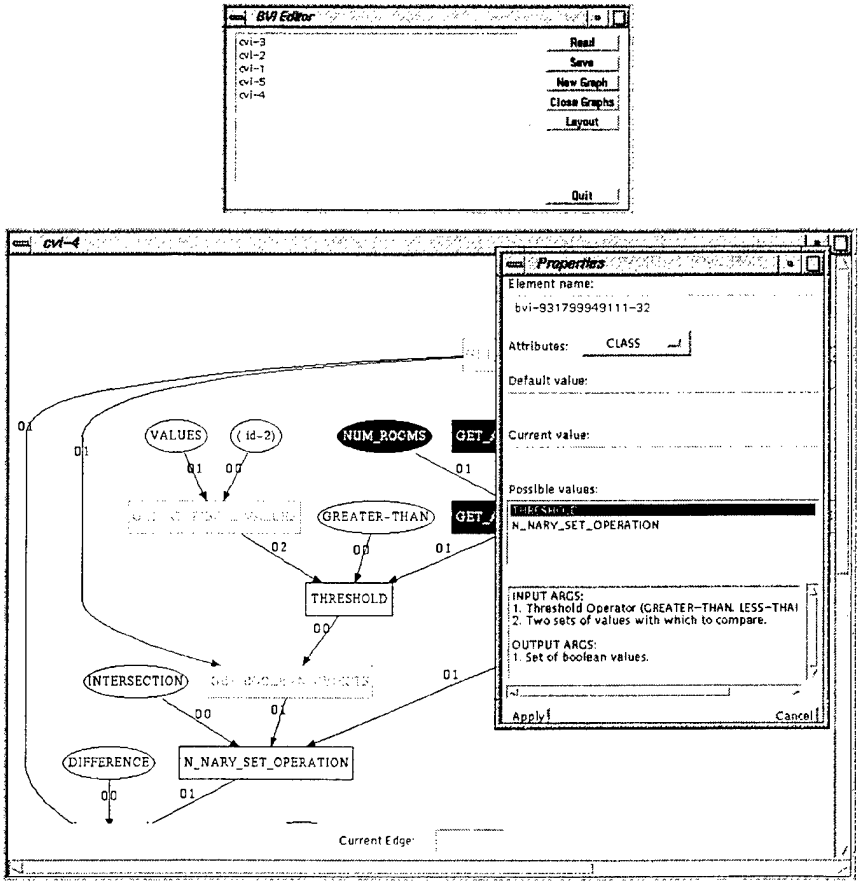


Figure D-15: Functional specification editor

By using this editor we generated all the example visualization techniques shown in chapters II and III including *painting* [*Becker, 1987*], *dynamic queries* [*Ahlberg, 1992*], *aggregation, Visage drag-and-drop* [*Roth, 1996*], *semantic zooming* [*Bederson, 1994*], etc. These examples illustrate the wide range of visualization techniques that may be captured by our framework and created using our functional editor. In chapter III-2, we used this

interface to customize the *dynamic query slider* interactive technique and change its behaviors and effects. Each change is simple and time effective, showing how our visualization functions framework may simplify the customization, and prototyping of interactive visualization interfaces. By building these examples with our editor we show that our visualization functions framework is not merely a theoretical description of visualization techniques, but is complete and specific enough to be implemented and realized as active systems for creating and prototyping visualization behaviors and metaphors. Depending on the complexity of the design and the amount of structure sharing that is possible with previous techniques, creating each of these examples may take from 20 minutes up to several hours. In either case, the time taken is still insignificant compared to writing code from scratch. The editor interface can be further improved by considering the specification by example methods suggested by Myers [*Myers, 1991*].

Appendix E : Evaluation

Using GOMS to Evaluate our Automatic Design System

In chapters II and III we presented a framework for characterizing visualization techniques. This framework presented four classes of visualization primitives: data, mapping, graphical and rendering. It has been shown in many current hand-made visualization systems that utilizing all four classes of these primitives can significantly improve our ability to solve tasks and communicate information with visualizations. Previous work in automatic visualization design, however, only considered the use of mapping techniques. In chapter IV we outlined a set of metrics and heuristics that enable data techniques to be added into the automatic design process. These heuristics describe how data and mapping techniques can be successfully combined and effectively traded-off to best solve tasks. In appendix F we describe how graphical and rendering techniques might also be integrated into the automatic design process.

To evaluate the completeness, generality and practicality of our framework as well as the effectiveness of our metrics and heuristics, we used the theories and concepts developed in chapter II, II, and IV to implement an automatic design system (AVID – Automatic Visualization Interface Designer). We describe the implementation of AVID in detail in chapter V. Here we evaluate the results of AVID using GOMS [*Card, 1980*]. Specifically we want to test the following:

- Our theories can be implemented and they perform as expected. I.e. the design metrics and heuristics used in the designer result in output designs that are ordered according to complexity of use (by "complexity of use" we refer to cognitive, perceptual, and motoric complexity).
- Our work increases the breadth of designs that can be generated by automatic systems. I.e. our automatic system should be able to produce designs that cannot be previously generated.
- Our work improves the effectiveness of visualizations generated. I.e. the expanded design space contains visualizations that allow certain task classes to be solved more effectively.

To demonstrate the effectiveness of the expanded design space, we have chosen three tasks that span the three major classes of Exploratory Data Analysis (EDA) tasks as captured by previous work, and which we find interesting in our own work. Specifically, the three classes of tasks are: search, compute, and comparison. For each of these tasks we used our automatic system (AVID) to generate up to about 20 designs, exploring a maximum of 15000 nodes in the design space (i.e. 15000 different design states). A design state can be differentiated from all others either because the graphical elements used are different, the data encoded within the design is different, or the constraints (graphical or data) placed on the design are different.

For each task, we chose from the 20 designs, a set of about 7 or 8 that had interesting design differences and performed a GOMS analysis on each of them to estimate the total time taken by experts to perform the intended tasks. Specifically, for each task and each design we developed a procedure or an algorithm that might be executed by an **expert user** to solve the task. Implicit in these procedures are many assumptions about the cognitive, perceptual, and motoric steps executed by the user. There are obviously many different correct algorithms that may be used and a complete analysis would try to model all of them. For our purposes however, we chose a single straightforward efficient algorithm for each layout assuming **complete understanding of the graphic and task** (i.e. no time to allocated for interpreting the graphic and recalling the task). Taking the difficulty of representational interpretation into account during the automatic design process would be an interesting area of study for future work. Based on this "complete understanding" assumption, we presuppose the following rules in **all** of our GOMS evaluations:

Baseline Assumptions for ALL tasks and ALL designs

1. The user has the task committed to memory and the task does not change during the experiment.
2. The user has complete understanding of all graphic designs used and thus we **do not** account for time taken to understand a graphic (i.e. time taken to parse what data attributes are shown, what data attributes are mapped to which graphical properties, etc).

Generating a single procedure for each design is sufficient for our purposes because our goals are to contrast the different visualizations produced by our system and get some general time estimates for determining the correctness of its design ranking. For this purpose we do not need very accurate total task time measurements. It is sufficient that we determine general design groups based on approximate total time differences and based on these groups ensure that our system does indeed assign meaningful costs to its output visualizations.

We expect that this simplification will cause our models to under predict average performance time because not all users are likely to adopt efficient algorithms. It is also important to note that our procedures are based on a single data set, i.e. the data set used to generate the visualization designs. Different data sets will invariably cause different GOMS sequences and total times to be generated. Depending on how important the data distributions are to the chosen procedure, a change in data set may cause a simple multiplicative change to the total time based on number of data elements, or it may cause significant changes to the actual procedure used to solve the task. We have discussed in appendix C-4.3, how our automatic design system takes some of these data size and data distribution effects into account. A more complete treatment of these issues however is left for future work.

Most of the GOMS operators and estimated times used in our analysis are taken from Lohse et al. [*Lohse, 1993*] and John et al.[*John, 1990*]. A summary of these operators are shown below in Table E-1.

Lohse and John in turn based many of their operator times on previous empirical work as is shown in the *original source* column of Table E-1.

GOMS operator	Estimated time (msec)	Original source	Explanation
Perceive simple binary signal	100	Card et al., 1983 [*Card, 1983*]	
Perceive complex visual signal (word or code)	290	John & Newell, 1989 [*John, 1989*]	As was done in *John, 1990*, we divide the 340 msec estimated time by *John & Newell, 1989* into a perceive component (290 msec) and a verify component (which is a cognition operation taking 50 msec). This change makes a difference when processing multi-word entries where only a single verification is needed at the end of the entry. This 290 msec time also conforms well to the time listed by *Lohse, 1993*.
Cognition (mental step)	50	John & Newell, 1989 [*John, 1989*]	As was listed by *Lohse, 1993*, *Cavanaugh 1972*, *Olson, 1990*, and *Welform, 1973*, all measured various cognitive time estimation to perform a mental step, or to compare various types of objects in working memory. These empirical times range from 33 msec to 92 msec. The average time of all these operations came up to 55 msec. This figure is similar to the 50 msec time estimated by *John & Newell 1989* which we will adopt in our GOMS sequences.
Eye saccade (travel time)	30	John & Newell, 1990 [*John, 1990*]	*Card 1983*, estimated eye movement times to be 230 msec. This figure was later refined in *John & Newell 1990* and divided up into smaller steps taking into account the time needed to initiate the eye movement, the actual travel time, and the fixation time. In our analysis we will use the latter model for better time estimation accuracy.
Homing between devices	350	Card et al., 1983 [*Card, 1983*]	In general, *Card 1983* estimated that the time taken to move between devices is 350 msec. In our analysis however, the only devices that the user needs to move between include the numeric keypad and the mouse. Since both these devices are commonly placed in close spatial proximity (assuming right handed mouse use), the time taken to move between the two devices will likely be smaller. Using Fitts Law, we estimated the time taken for the move to be approximately 132 msec (assuming 6 inch distance between mouse and numeric keypad, and 3 inch width for the numeric keypad).
Hand movement between numeric keypad and mouse	132	derived from Fitts Law [*Card, 1983*]	
Horizontal movement within numeric keys	40	derived from [*John, 1988*]	*Lohse, 1993* estimated a keypunch entry to take 372 msec, however in our analysis we decided to break up keypunch entries into smaller steps including upstrokes, downstrokes and finger movements as was done in *John, 1990*. This will allow us to more accurately model time savings for numbers with many repetitious digits.
Upstroke	60	[*John, 1988*]	
Down-stroke	60	[*John, 1988*]	

Table E-1: Summary of all GOMS operators used in the evaluation sequences listed in this appendix

The results of all the GOMS analyses for each of the three EDA tasks (search, computation, comparison) are shown in subsequent sections. Essentially our evaluation test designs and GOMS estimates show the following:

- The output order of our automatic designer (i.e. our design heuristics used) does indeed conform to cognitive, perceptual, and motoric complexity as computed by GOMS. I.e. the theoretical concepts developed here for characterizing and expanding the visualization techniques design space for automatic visualization generation can be implemented and the results are meaningful (i.e. conforms to GOMS computed times).
- Adding data transform techniques into the automatic design process expands the visualization design space and enables whole new sets of interactive and non-interactive visualizations to be generated. Many of these designs are shown in the following sections.
- Some of the new designs generated as a result of work developed in this thesis (i.e. pure data transform techniques and hybrid data + mapping designs) perform much better than the designs that can be generated with current state of the art technology (i.e. pure mapping designs) for the task classes we considered (*computation, search, comparison*); with the highest gain in computation tasks.

In section E-5 we outline our reasons for using GOMS as an evaluation method and describe why it can be appropriately applied for our purposes.

E-1 Task 1: Find Task

Finding a "Good" University based on Graduation Rates and Test Scores

Suppose we are looking for a "good" university to attend. Some of the attributes we may be interested in include the quality of students attending the school (which can be deduced by looking at their ACT and SAT scores) as well as whether the school has a good track record for graduating its students. The task specification entered into our automatic designer is shown below:

```
(lookup
  (and-obj
   (find-obj '(VALUE . GREATER-THAN)
             (lookup '(OBJECT . nil) '(VALUE . AVG_COMBINED_SAT))
             '(VALUE . ?))
   (find-obj '(VALUE . GREATER-THAN)
             (lookup '(OBJECT . nil) '(VALUE . AVG_ACT))
             '(VALUE . ?))

   (find-obj '(VALUE . GREATER-THAN)
             (lookup '(OBJECT . nil) '(VALUE . GRADUATION_RATE))
             '(VALUE . ?))
   )
  '(VALUE . OBJECT-NAME)
  )
```

Task E-1: Search for universities based on the average SAT and ACT scores of attending students as well as graduation rates.

From analyzing the set of visualization displays generated by our system, we were able to determine the schools that performed well based on the three search attributes: *Duke, Emory, Vanderbilt, University of Pennsylvania, Massachusetts Institute of Technology,* and *Brown University*. It is perhaps not too surprising that all of these universities are private schools.

The task specification entered into our designer provides for some amount of flexibility in the search parameters. The wild card "?" symbols indicate to the designer that we are not sure of the exact *scores* and *graduation rate* thresholds of interest in our search, so that the designs generated will allow users to experiment with different search thresholds. In the interest of simplicity, however, we only estimate the time for setting one set of thresholds in the GOMS sequences below. Increasing the number of different threshold conditions tested will not change the general GOMS procedures used, but merely result in a higher time multiplicative cost commensurate to the number of different threshold tests.

In the next sections we present a set of seven output designs generated by our designer, ordered from best (least system assigned cost) to worst (highest system assigned cost). Each design is accompanied with a description of the visualization as well as a GOMS evaluation showing the steps taken to complete the above task and the total time taken. In the summary section we compare the GOMS estimated times for all seven designs and provide an analysis of the results and its impact on our goals.

In all following GOMS analysis for this task we assume:

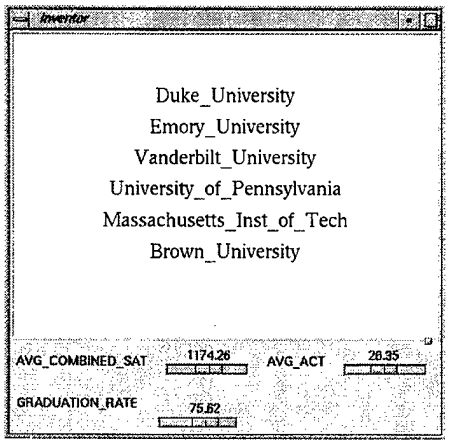
Baseline Assumptions for Task 1 (Search Task):

1. Only one set of threshold conditions is used. In tasks of this type it is common for users to experiment with different search constraints until a satisfying number of universities are retrieved. Setting overly high threshold values may cause no universities to fit the conditions of the search while setting threshold values that are too low may cause too many universities to fulfill our search criteria. However, to model the actual number of task conditions a user would actually try out in a session, we would need to account for user preferences and experience, the difficulty of the interface, the importance of the task, etc. Such modeling, if possible, is beyond the capabilities of the analysis techniques used here. As a result, all the GOMS sequences for this task only accounts for one set of search conditions. We will however show how experimenting with multiple search conditions can change the total estimated time for each design.
2. Assume that the user's hand is already on the mouse at the start of the task. We believe that this is a reasonable assumption since the user is most likely already working or interacting with the computer when the designs are shown to them.
3. We assume that the user has some approximate threshold values in mind for each of the three attributes. To get to each of the threshold values, the user must either make multiple moves on a slider

input device, or scan around the different encoding positional axes. In general we assume that three adjustments are needed to get to the correct value on a slider or to find the desired value on a positional axis. This figure is based on our own experiences in using the generated interfaces. We want to point out, however, that since both operations have comparable time estimates (320 + 490*n* msec for manipulating a slider and 470*n* for an axis lookup, where *n* is the number of adjustments needed), there isn't much of a time effect if this number (*n*) turns out to be slightly higher or lower. We anticipate that this number may range from about 1 to 20 depending on the adeptness of the user in manipulating devices (physical or virtual), the sensitivity of the slider, the length of the slider, the accuracy of manipulating the slider, the task accuracy required, the task importance etc. Such variation however does not cause significant time changes among the designs. When $n = 20$, the estimated time difference is 400 msec. only.

E-1.1 Design 1

In this design the search results are pre-computed by the system and only the universities that pass all of the search constraints are shown. The visualization interface generated by our automatic designer provides three sliders, each allowing the user to input a threshold constraint for each of the search attributes (*avg_combined_SAT, avg_ACT,* and *graduation_rate*). A big advantage of this design is the reduction of visual clutter due to the fact that the search data need not be shown to users, as their results have been pre-processed by the system. The nature of the search task allows even greater visual benefits because we can prune the number of objects in the display so that only the relevant ones are shown.



To solve the task using this design users must set the sliders to the relevant threshold values and then read off the university names. The time taken for this design is mostly attributed to the effort of manipulating the sliders. We assume that to get the desired threshold value on a slider, the user needs to adjust the slider an average of three times.

General goal	Cognitive, perceptual, or articulatory step taken by user		Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Alter first slider to required SAT threshold score	Attend alter slider		50			
	Initiate finger drop		50			
	Finger drop		60			
	Sub-total			160		
Shift the slider until the correct threshold value appears. Note that here, the process of performing the finger moves necessary to complete the task is performed in tandem with scanning the slider to determine the current threshold value. In this case since the hand move required is minute, the time is dominated by the scan and reading operation.	Attend hand move		50			approx. 1180 for SAT approx. 26.5 for ACT approx. 76% for graduation rate
	Initiate hand move		50			
	Hand move	Attend check the correctness of current threshold	50			
		Read threshold	290			
		Verify results	50			
	Sub-total			490		
As was previously noted, we assume that three slider moves are necessary to get to the desired threshold value.	3 * 490				1470	
Finish manipulating slider	Attend finger lift		50			
	Initiate finger lift		50			
	Finger lift		60			
	Sub-total			160		
Total time for manipulating ONE slider	160 + 1470 + 160		1790			
Total time to manipulate all sliders	3 * 1790				**5370**	
Read university names. Here we assume that there are a limited number of names (six), thus the user can get to them with a limited number eye movements. We assume the user can get to three names with one eye movement based on the required font size for reading.	Attend name read		50			1. Duke, 2. Emory, 3. Vanderbilt, 4. University of Pennsylvania, 5. Massachusetts Institute of Technology, 6. Brown University.
	Initiate eye movement		50			
	Eye movement		30			
	Read 3 university names with info. verification 3 * (3 * (290 + 50))		3060			
	Initiate eye movement		50			
	Eye movement		30			
	Read 3 university names		3060			
	Sub-total			6330	**6330**	
Total time	5370 + 6330				**11700**	

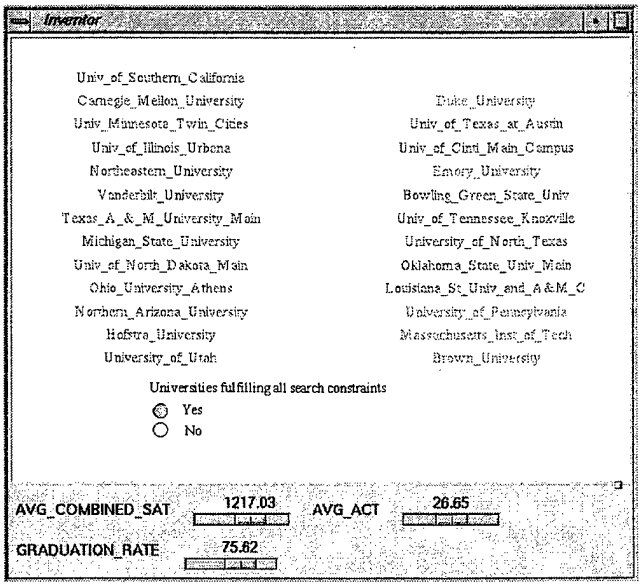
It is important to note that a major part of the total time in the design above (approximately half) can be attributed to the time taken to manipulate the three sliders. This component increases the more sets of search conditions we want to test. Each change in condition increases the total time by 1790 msec for manipulating a slider and an additional 280 msec for a quick scan of the results to see whether we have the approximate number of data concepts desired (as is shown below). Thus total time for each search condition change is 2070 msec. Note that this figure assumes that three minute adjustments are needed to get to the desired threshold value on the slider. As a user gets more familiar with the slider mappings and

sensitivity, this figure may be reduced at which time the cost of each change may only take 810 msec + 280 msec = 1090 msec (assuming that no adjustments are needed). Also note that when scanning for the number of elements retrieved, we assume that no counting is required because we are not interested in the exact number of elements, just in whether the approximate number of elements retrieved falls within our task requirements.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Quick scan of results	Attend see approximate number of objects returned by search	50			
	Initiate eye movement	30			
	Perceive results (This is a pre-attentive operation)	100			
	Compare general size with desired size	50			
	Verify results	50			
	Sub-total		280		

E-1.2 Design 2

This design is very similar to the previous one in that the search results are pre-computed. However, instead of filtering the visualization display to only show the universities that fulfill the search, we map the search results to *hue (color)*. *Blue* universities indicate those that pass the search and *red* universities indicate those that don't. Because more data concepts are shown in this design, and an additional graphical property (*color* or *hue*) is used to show the search results, this design is rated lower than the previous one.



The GOMS procedure used for this design is identical to the previous one, except in the last portion when we are reading the university names. Rather than just reading the names top down as in the previous example we need to move our eye to each *blue* colored university before reading their names. This results in a slightly higher time cost for processing the end results.

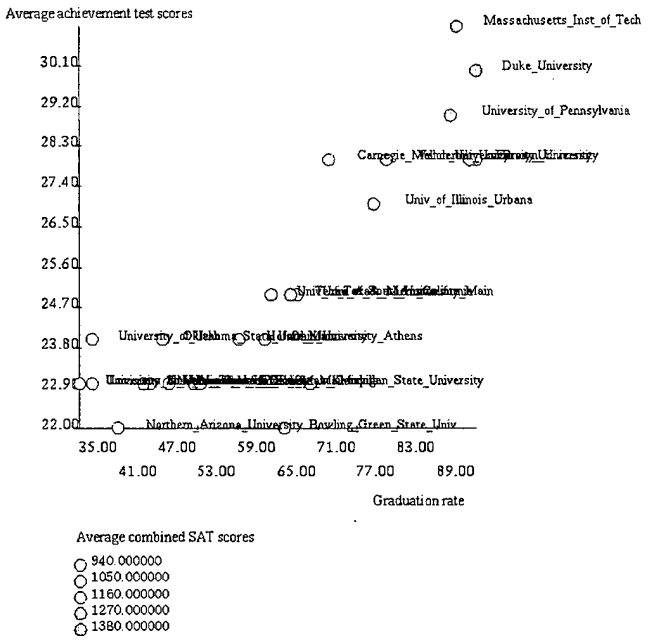
General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Total time taken for slider manipulation (taken from time calculated in Design 1)		5370		**5370**	approx. 1180 for SAT approx. 26.5 for ACT approx. 76% for graduation rate
Scan to first blue label	Attend get next blue label	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive label	100			
	Verify label is blue	50			
	Sub-total		280		
Read label name	Attend label read	50			1. Duke, 2. Emory, 3. Vanderbilt, 4. University of Pennsylvania, 5. Massachusetts Institute of Technology, 6. Brown University.
	Read university name	870			
	Verify results	50			
	Sub-total		970		
Total time to process each blue label	970 + 280	1250			
Repeat for each blue colored label	6 * 1250			**7500**	
Total time	5370 + 7500			**12870**	

An advantage of this design is that all of the universities are shown, thus users will not get disoriented from changes in the number and positions of the universities caused by changing search conditions as in the previous case. In most cases however, the effect of such disorientation is minimal, and it can only become significant when we are trying to track particular data concepts across many different search conditions. Because the slider manipulation portion of this design is identical to the previous one, the cost of experimenting with different search conditions is also identical at 1790 msec or 1090 msec. for experts.

E-1.3 Design 3

This design unlike the two previous ones has a completely perceptual design, i.e. all of the task data are mapped to graphics and it is up to the user to perform the search perceptually. Each mark cluster represents a university. *Avg_combined_SAT* score is mapped to mark *saturation*, *avg_ACT* score is mapped

to *y-position*, *graduation_rate* is mapped to mark *x-position*, and *university_name* is mapped to a *text label* next to each mark. An advantage of this design is that no input device manipulation is required. A weakness is the additional load of performing the search perceptually. In addition, more data has to be mapped and this might sometimes cause readability problems, as is the case below where some of the objects are occluded. Currently our automatic designer is unable to provide solutions to such readability issues but refer to appendix F to see how such additions might be integrated into our automatic design system in the future. Our designer assigned a higher cost to this visualization because of the two reasons stated above: additional perceptual load and additional visual clutter.



To solve the search task using this design, we first refer to the *saturation* legend to determine the appropriate saturation color for our desired *avg_combined_SAT* threshold. Since there are only 5 gradations to the saturation legend, we can zero in on the last two values in the legend and interpolate between them (i.e. only two values need to be processed). After processing the saturation legend we look to each of the positional axes to determine the position of the *avg_ACT* score and *graduation_rate* thresholds that we are interested in for the search. As with the case of the slider adjustments, we assume that the user will not be able to get to the exact position on an axis with the desired threshold value on the first try, but an average of three eye movements are required. Once the positions are determined, we can perceptually demarcate an area containing all the universities that fit our *avg_ACT* score and *graduation_rate* criteria. Based on our task we are interested in high *avg_ACT* score and *graduation_rate* and this is captured by objects in the upper-right hand corner of the chart. From all the marks in this area we choose the ones with greater *saturation* values than our desired threshold (which we determined at the start of the procedure). When we have identified the relevant marks, we read off the associated university name that is encoded next to it.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Lookup *saturation* legend for desired *SAT score saturation* value.	Attend legend lookup	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read last value	290			
	Verify value	50			
	Read previous value	290			
	Verify value	50			
	Attend perform saturation interpolation based on values	50			
	Perceive saturation values	100			
	Interpolate	50			
	Verify saturation	50			
	Sub-total		1060	1060	
Note that in this legend lookup step we assume that the user is able to keep the determined saturation value and use it when it is required later. Also note that since the saturation legend scale only has five gradations, it is probably sufficient for the user to zero in on the last mark (most saturated mark) and do some slight interpolation estimate of saturation value with the previous mark to get the general saturation level required. As a result no adjustments are needed to get to the desired saturation threshold, as is the case with the two axis positions next.					
Scan to desired *graduation rate* on *x-axis*	Attend scan	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read value	290			
	Verify value	50			
	Sub-total		470		
Repeat 3 times to get to desired value	(3 * 470)	1410		1410	
Scan to desired *ACT score* on *y-axis* with 3 repetitions to get desired value		1410		1410	
Scan to xy-position which corresponds to desired *graduation rate* and *ACT score*	Attend scan	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive general area	100			
	Sub-total		230	230	
Scan for marks with the requisite saturation values as determined in a previous step. This is a pre-attentive operation thus users should be able to zero in on the relevant saturation without having to attend to each mark	Attend scan for marks with requisite saturation	50			MIT, Illinois, Pennsylvania, Duke (Emory, Vanderbilt, and Brown are occluded)
	Initiate eye movement	50			
	Eye movement	30			
	Perceive mark saturation	100			
	Verify mark saturation	50			
	Attend university name read	50			
	Read university name (3 * 290)	870			
	Verify university name	50			
			1250		
Repeat 7 times for each relevant university. Note that in actuality there are only 6 universities that pass all search conditions but because of saturation inaccuracies 7 are found instead. (assume for here that there is no	(7 * 1250)	8750		8750	1. Duke, 2. Illinois, 3. Emory, 4. Vanderbilt, 5. University of Pennsylvania, 6. MIT,

occlusion so all labels can be read)					7. Brown University.
Total time	1060 + 1410 + 1410 + 230 + 8750			**12860**	

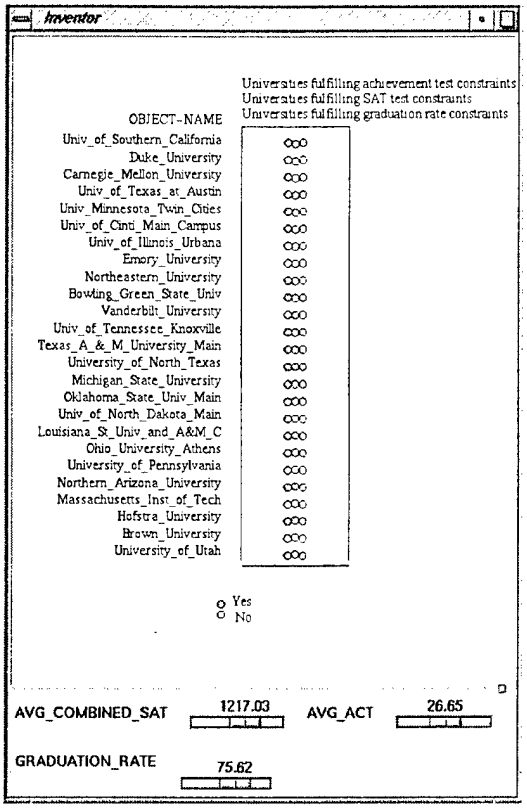
It is interesting to note that this design actually has a very slightly lower GOMS estimated total time compared to Design 2. However it should be pointed out that the GOMS estimation does not take into account the fact that there is occlusion in the display and the task **cannot** be fully completed using the given design. In particular *Emory, Vanderbilt* and *Brown* are occluded and cannot be extracted from the visualization display. A better label placement algorithm would improve this situation. With interactive enhancements, it is also possible to solve this problem as with labels on demand [*Plaisant, 1996*], however, these enhancements will require input device manipulation and this will result in increases in the total time taken. Another important issue here is the fact that saturation is not a very accurate encoding property and as a result mistakes may be made in identifying universities with the required *avg_combined_SAT* score (which is the data attribute encoded with saturation). An example in this design is *University of Illinois* which passes both the *avg_ACT* and *graduation_rate* conditions but not the *avg_combined_SAT* condition. However because of the lack of accuracy in saturation, it is incorrectly chosen as an acceptable search candidate.

It should be noted however that the time taken for task condition adjustments here are negligible. This is because once the general search area (i.e. upper right hand corner) is located and the general required saturation values are noted, minor adjustments in the search would only involve some small adjustments to the current search area or acceptable saturation values and we do not even need to consult the axes or legends. In contrast, even small changes in search conditions require the same slider movements in the previous designs. Thus in cases where the data set is small (not high probability of readability problems) but the search conditions need to be changed often, a purely perceptual design is probably superior to the pre-computed ones. In fact as was shown in appendix C-4.2.2, in search tasks that require less information to be shown or fewer search attributes, our designer picked the perceptual design over the data computed design because of the additional input device manipulation load associated with the latter.

E-1.4 Design 4

This design is a hybrid design where each of the search conditions in the task are computed by the system, but the *AND* task is left to be performed perceptually by the user. Each university concept therefore has three marks associated with it, indicating whether it passed each of the search conditions. A *blue* mark indicates that a search condition was fulfilled while a *red* mark indicates that a search condition was **not** fulfilled. Similar to Design 1 and Design 2, the interface has a set of sliders through which users may use to set the current search condition thresholds. This design is ranked lower than the previous designs because unlike Design 1 and Design 2 which only shows the university names, this design shows four objects, the

university names together with a cluster of three marks representing each of the search conditions. The left-most mark represents the *graduation_rate* search condition, the middle mark represents the *avg_combined_SAT* score search condition, and finally the right-most mark represents the *avg_ACT* score condition. In addition to the increase in perceptual complexity of the graphic, the perceptual load is also increased because now users must deduce the results of the *AND* task perceptually. However, the advantage of this design is that each of the search conditions are shown, and in cases where the user must trade off one condition against another, it is useful to have each of the search condition results available for perusal. If we compare Design 3 and Design 4, our system gave preference to Design 3 because while there is slightly less perceptual complexity here (.i.e. fewer different graphical properties are used and the results are accurate), there is the added cost of slider manipulation.



To solve the search task using this design, we must initially put the sliders to the desired threshold values (similar to Design 1 and Design 2). Once we have done this, we can find all relevant universities by scanning down the table and picking universities that have a three-mark cluster that is all *blue*. While scanning for *blue* objects is pre-attentive, scan for rows with **all** *blue* objects is not. Therefore we assume that the user must visit each row that contains at least one blue object. In the example above, this only increases the number of rows visited by two. However, depending on changing task conditions and the data distribution, this figure may increase.

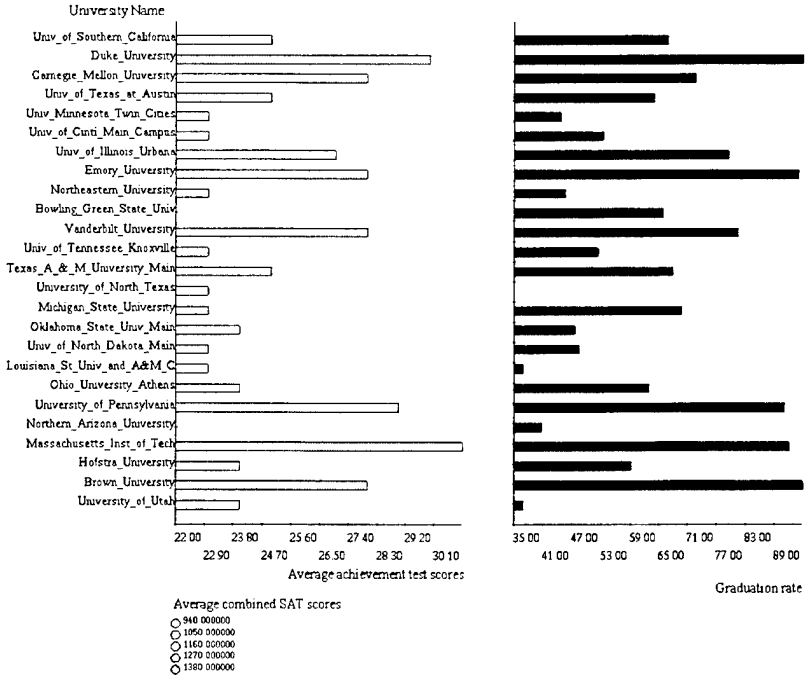
General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Total time taken for slider manipulation				**5370**	approx. 1180 for SAT approx. 26.5 for ACT approx. 76% for graduation rate
Scan to next row with blue in it, and verify that all three marks are blue	Attend get next row with blue objects	50			1. Duke, 2. CMU, 3. Illinois, 4. Emory, 5. Vanderbilt, 6. University of Pennsylvania, 7. Massachusetts Institute of Technology, 8. Brown University.
	Initiate eye movement	50			
	Eye movement	30			
	Perceive results	100			
	Verify all three marks are blue	50			
	Sub-total		280		
Repeat for each row with blue in it	8 * 280	2240		**2240**	
If row contains all blue marks, scan to left to read university name.	Attend read university name	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read university name (3 * 290)	870			
	Verify results	50			
	Sub-total		1050		
There are 6 rows with three blue marks	6 * 1050	6300		**6300**	
Total time	5370 + 2240 + 6300			**13910**	

We would like to point out that although the total estimated time for this design is fairly close to those of previous designs, there is a significantly higher cost associated with changing the search conditions. This is because with this design, it is more difficult to quickly determine the number of universities that actually pass all search conditions. In all previous cases this operation is fully pre-attentive, and users can get a general feel for the size of the search results by just looking once at the visualization display as a whole. In contrast, this design requires users to attend to each row with blue objects in it to determine whether it actually passes all search conditions. Specifically, even though there may be a lot of *blue* dots in the display (which we can determine pre-attentively) it does not mean that many universities passed all search conditions (i.e. there may be many rows with one or two *blue* marks but not three). To determine the actual number of objects that fulfill the search, we need to attend to each row containing *blue* objects. Thus the cost of each search condition change is 1790 msec for the slider manipulation + 280*n* msec for processing each *blue* row (where *n* represents the number of rows with *blue* in them). For larger *n*, the time taken could become very significant. We also want to point out that when there are many *blue* dots in the display, an effective alternative for finding concepts that fulfill all search conditions might be to look for rows with *red* in them, and then just subtract the total number of rows from the number of rows with *red* dots (i.e. concepts that did not pass at least one search condition). Thus *n* is bounded by half the size of the entire

data set. Another alternative strategy is to only visually process the column of marks that correspond to the changing condition, and then only attend to those rows where there are changes. This strategy however results in approximately the same time estimation because each candidate row has to be attended to individually.

E-1.5 Design 5

This design is a purely perceptual design similar to Design 3. Unlike Design 3 however, the three search attributes are spread over two different charts. *University name* is mapped on the *y-axis*, *avg_combined_SAT* score is mapped to bar *saturation* on the left chart, *avg_ACT* score is mapped to *x-length* on the left chart and *graduation_rate* is mapped to *x-length* on the right chart. One notable difference between this design and Design 3 is that here, there is no occlusion among the elements because now the information is separated over a greater number of spaces. While not having occlusion problems is a big plus, this less integrated design also increases the perceptual load and subsequently the total time taken to complete the task.



The GOMS procedure for this design begins with a lookup on the *saturation* legend to determine the saturation value corresponding to our desired *avg_combined_SAT* threshold, similar to what was done in Design 3. Next, we begin with the left chart (assuming the Western convention of reading from left to right) and lookup the appropriate *avg_ACT* threshold value on the *x-axis* and then proceed to the right chart to lookup the appropriate *graduation_rate* threshold there as well. Once we have determined the *x-positions* of our desired threshold values, we begin scanning up the display, and processing any bar that intersects our scan line. For each of these bars, we first check their *saturation* value to ensure that they do indeed pass our *avg_combined_SAT* threshold value. If they do, we scan to the right chart and determine whether the

associated *graduation_rate* passes our desired threshold *x-position*. We do this by looking to the rightmost end of the bar and then comparing that with our remembered threshold position for *graduation_rate*.

Note that an important consideration for the GOMS procedure of this task is the number of times the axes of the two charts must be consulted to get the proper *avg_ACT* and *graduation_rate* search threshold positions. In Design 3, we assumed that the axes only had to be referred to once in the entire data analysis process and the user is able to maintain the general search area and *saturation* in STM (short term memory). However, this assumption no longer holds true here because the information is separated over multiple spaces, and when processing each space we lose our positional context in the other space. Since we must switch from space to space when processing each university row it is difficult to accurately maintain the search threshold positions in either of the spaces. As a result several references may need to be made to the axes as a reminder of the search threshold positions. In the GOMS procedure below we assume that references are made to the axes for those bars that are relatively close to the desired threshold positions. This is because when bars greatly exceed or do not meet the positional thresholds, users can tell whether a search condition was met without having to accurately remember the exact search positions.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Lookup *saturation* legend for desired *SAT score saturation* value. (computed in Design 3)		1060		1060	
Scan to desired *graduation_rate* on *x-axis* of right chart with 3 repetitions to get desired value. (computed in Design 3)		1410		1410	
Scan to desired *ACT score* on *x-axis* with 3 repetitions to get desired value. (computed in Design 3)		1410		1410	
Scan in left chart and determine the next bar above the *avg_ACT* threshold value (assume that approximate threshold position can be kept in STM)	Attend determine if left bar length passes threshold position	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive rightmost edge	100			
	Verify if greater than threshold *avg_ACT score* position	50			
	Sub-total		280		
Check that saturation value passes desired *avg_combined_SAT* threshold.	Attend check saturation	50			
	Compare saturation	50			
	Verify results	50			
	Sub-total		150		

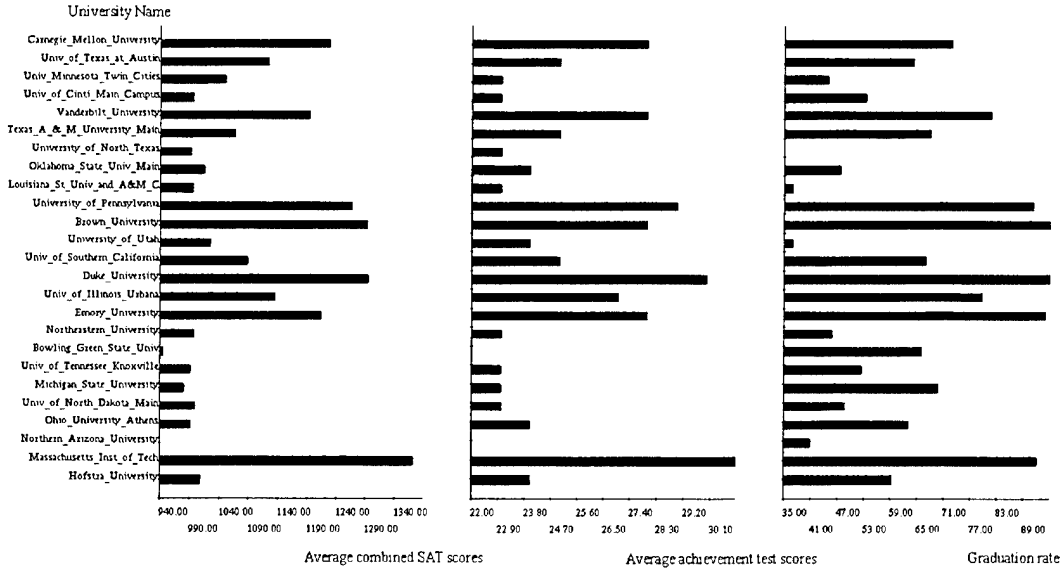
Scan to rightmost edge of bar on the right chart in the current row	Attend determine if right bar length passes threshold position	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive rightmost edge	100			
	Verify if greater than threshold *graduation_rate* position	50			
	Sub-total		280		
Total time to process one bar row (assuming no references to axes)	(280 + 150 + 280)	710			
Repeat for each candidate bar row	8 * 710	5680		5680	1. Duke, 2. CMU, 3. Illinois, 4. Emory, 5. Vanderbilt, 6. University of Pennsylvania, 7. Massachusetts Institute of Technology, 8. Brown University.
Time taken for axes references: Note that axes references are needed for bars in the first and second charts that are considered in the above process, and that fall close to the *x-position* threshold positions of their respective charts.					
Time taken for each axis reference	Attend axis reference	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read closest label	290			
	Compare label with desired search threshold	50			
	Verify if axis passes the desired search threshold.	50			
	Sub-total		520		
Three axis references are required in the left chart	3 * 520	1560			Brown, Vanderbilt, Illinois
Three axis references are required in the right chart	3 * 520	1560			Vanderbilt, Illinois, CMU
Total time for axes references	1560 + 1560			3120	
For each university row that fulfills all search conditions, scan left and read the relevant university name. There are in actuality 6 universities that pass all search conditions, however in this design 7 universities are found, including *University of Illinois* because of inaccuracies in the saturation encoding.					
Scan to left to get single university name	Attend get university name	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read university name (3 * 290)	870			
	Verify name	50			
	Sub-total		1050		
Repeat for each university that passed all conditions	7 * 1050			7350	1. Duke, 2. Illinois, 3. Emory,

					4. Vanderbilt, 5. University of Pennsylvania, 6. Massachusetts Institute of Technology, 7. Brown University.
Total time	1060 + 1410 + 1410 + 5680 + 3120 + 7350			**20030**	

As can be seen from the GOMS procedure, the lack of integration in this design, in addition to the loss in accuracy from the *saturation* encoding resulted in a greater total processing time compared to the previous examples. The cost of changing the search conditions in this design is also relatively large. This is because any change in condition requires new candidate rows to be processed for acceptability or current rows to be re-processed for possible rejection. As was calculated in the GOMS table above, processing each row takes approximately 710 msec without including time for the positional axes reference. Each change in condition may require any number of rows to be processed depending on the size of the change. Even with a processing cost as low as two to three rows per change, the time taken exceeds that of previous designs. This is because unlike Design 1, Design 2, and Design 3, we cannot pre-attentively get a feel for the change in number of universities that fulfill our search conditions with each alteration. Also note that, as with Design 3 there are some inaccuracies here due to the *saturation* encoding of the *avg_combined_SAT* attribute. Specifically *University of Illinois* was chosen as an acceptable search candidate even though it failed to pass the required *avg_combined_SAT* threshold of 1180.

E-1.6 Design 6

Like the previous design, this visualization requires that the task be performed completely perceptually. However, the design separates out the search attributes into three charts, with each chart showing a single search attribute as bar lengths. The left-most chart shows *avg_combined_SAT* scores, the middle chart shows *avg_ACT* scores, and the right-most chart show *graduation_rate*. All three charts are aligned on their *y-axis* which encodes the *university names*. This design is rated lower by our designer specifically because of its extreme lack of data integration. As we have shown in the previous GOMS estimation, lack of integration can often cause significant additional perceptual loads from having to switch our attention from space to space, and keeping perceptual context during these switches. As we will show below this design is no exception. An interesting difference between this design and the previous one however is that *saturation* is no longer used to encode any attribute. As a result there are no errors stemming from our inability to interpret *saturation* values with great enough precision, as was the case previously.



The GOMS procedure for this design is very similar to the previous one except that here, instead of processing *saturation* we process an additional graphical chart. Processing again begins with getting the general threshold positions in each of the three charts. Once we have determined the threshold positions we process bars in each of the three charts, determining in each case whether the length is greater than our threshold position. Initially we look for a bar that passes the *avg_combined_SAT* condition in the first chart, then we proceed to subsequent charts to determine whether the other bars for the current university pass the other two conditions (*avg_ACT* and *graduation_rate*) as well. As was done previously, we will assume that axes references are only needed in those cases where the bar lengths are close to the search threshold positions.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Scan to desired *avg_combined_SAT* on *x-axis* of left-most chart with 3 repetitions to get desired value. (computed in Design 3)		1410		1410	
Scan to desired *ACT score* on *x-axis* of the middle chart with 3 repetitions to get desired value. (computed in Design 3)		1410		1410	
Scan to desired *graduation_rate* on *x-axis* of right-most chart with 3 repetitions to get desired value. (computed in Design 3)		1410		1410	

Now we begin processing each candidate bar row:					
Scan in left-most chart and determine the next bar above the *avg_SAT* threshold value (assume that approximate threshold position can be kept in STM)	Attend get next bar that passes the *avg_ACT* threshold	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive results	100			
	Verify results	50			
	Sub-total		280		
Scan to rightmost edge of bar on the middle chart in the current row and determine whether it passes the *avg_ACT* threshold value.	Attend determine if bar passes *avg_ACT* threshold	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive rightmost edge	100			
	Verify if greater than threshold *avg_ACT* position	50			
	Sub-total		280		
Scan to rightmost edge of bar on the right-most chart in the current row and determine whether it passes the *graduation_rate* threshold value.	Attend determine if bar passes *graduation_rate* threshold	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive rightmost edge	100			
	Verify if greater than threshold *graduation_rate* position	50			
	Sub-total		280		
Total time to process one bar row (assuming no references to axes)	(280 + 280 + 280)	840			
Repeat for each candidate bar row. Note that *University of Illinois* is considered a candidate in the first chart, but rejected right off, thus this additional processing adds 280 msec to the total processing time.	(7 * 840) + 280	6160		6160	1. Duke, 2. CMU, 3. Emory, 4. Vanderbilt, 5. University of Pennsylvania, 6. Massachusetts Institute of Technology, 7. Brown University.
Time taken for axes references: Note that axes references are needed for bars in any of the three charts that are considered in the above process, and that fall close to the *x-position* threshold positions of their respective charts.					
Time taken for a single axis reference	Time taken from computed value in Design 5.	520			
Three axis references are required in the left-most chart (Note that Illinois is rejected as a candidate here)	3 * 520	1560			Illinois, Emory, Vanderbilt
Two axis references are required in the left chart	2 * 520	1040			Brown, Vanderbilt
Two axis references are required in the right chart	2 * 520	1040			Vanderbilt, CMU

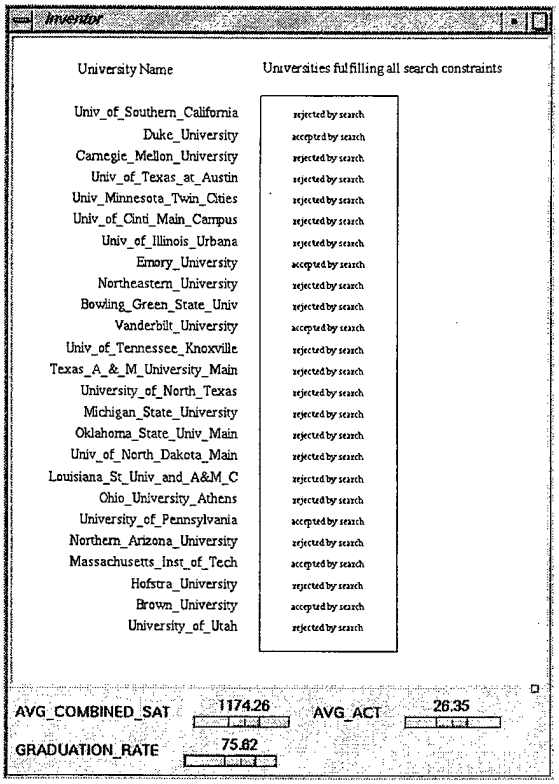
Total time for axes references	1560 + 1040 +1040			3640	
For each university row that fulfills all search conditions, scan left and read the relevant university name. There are 6 universities that pass all search conditions.					
Get university name	Attend get university name	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read university name (3 * 290)	870			
	Verify name	50			
	Sub-total		1050		
Repeat for each university that passed all conditions	6 * 1050			6300	1. Duke, 2. Emory, 3. Vanderbilt, 4. University of Pennsylvania, 5. Massachusetts Institute of Technology, 6. Brown University.
Total time	1410 + 1410 +1410 + 6160 + 3640 + 6300			**20330**	

It is very interesting to note that the total time estimated for this design is very close to that of the previous design. This is because whatever additional time required for processing the additional chart space is made up for in terms of the added accuracy from having all the search attributes mapped to position. As a result we do not need to process the *University of Illinois* element which was wrongly added as a search candidate in all previous perceptual designs that mapped *avg_combined_SAT* to *saturation*.

As with the previous case however, the cost of changing search conditions is high because it is difficult to estimate, for each set of conditions, the number of universities that are added to or rejected from the search. Any change in condition requires new candidate rows processed and as was calculated in the GOMS table above, processing each row takes approximately 840 msec (without including time for axes reference). This is a higher figure than even in the previous design.

E-1.7 Design 7

This design is interesting because it is a purely pre-computed design, that is ordered after a number of different purely perceptual and hybrid perceptual + pre-processed designs. The interface has a set of sliders for setting the three search threshold values and the results of the search are shown as *text labels* next to the university names. Universities that pass the search have the phrase "*accepted in search*" while universities that do not pass the search have the phrase "*rejected in search*" next to them. The reason this design is ranked so low by our designer is because the search results are shown as labels. This may make processing the search results extremely ineffective because now the operation is not clearly pre-attentive and users may need to process each university separately to determine whether it passed the search or not.



The GOMS procedure for this design includes all the slider manipulation operations that are identical to those required in Design 1 and Design 2. Once the search conditions are properly entered in, users must attend to each label to see whether a particular university passed the search or not.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Total time taken for slider manipulation	Time estimation taken from Design 1			**5370**	approx. 1180 for SAT approx. 26.5 for ACT approx. 76% for graduation rate
Now process each text entry:					
Scan to next entry and determine whether it is an "accepted" or "rejected" concept. Note that since the last two words of the two different phrases ("in search") are identical, it is only necessary to read the first word in the phrase.	Attend determine if next entry passed search	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read first word in phrase	290			
	Verify results	50			
	Sub-total		470		
Repeat for all entries	(25 * 470)			**11750**	

If entry is "accepted", scan to left and read university name	Attend get university name	50			1. Duke, 2. Emory, 3. Vanderbilt, 4. University of Pennsylvania, 5. Massachusetts Institute of Technology, 6. Brown University.
	Initiate eye movement	50			
	Eye movement	30			
	Read university name (3 * 290)	870			
	Verify name	50			
	Sub-total		1050		
Repeat for each "accepted" university	6 * 1050			**6300**	
Total time	5370 + 11750 + 6300			**23420**	

Note that depending on the phrase used to express the results of the search, it can be possible to view the results pre-attentively through *pattern/shape* matching or through *word length* matching.

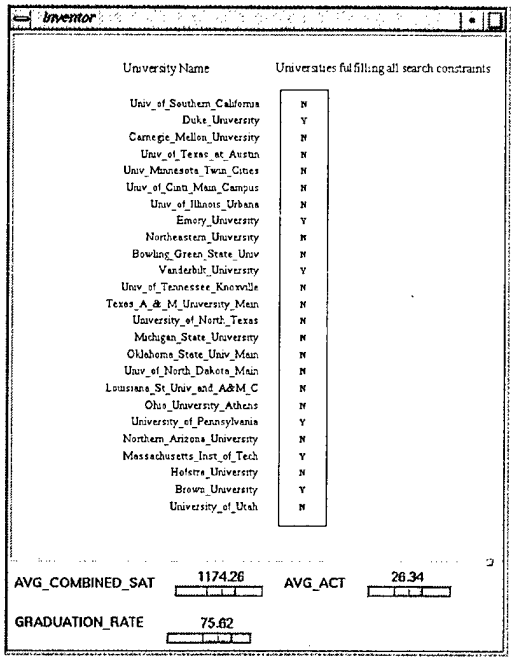


Figure E-1: Representing search results using text labels with "Y" or "N". Such results can be pre-attentively searched on through pattern matching.

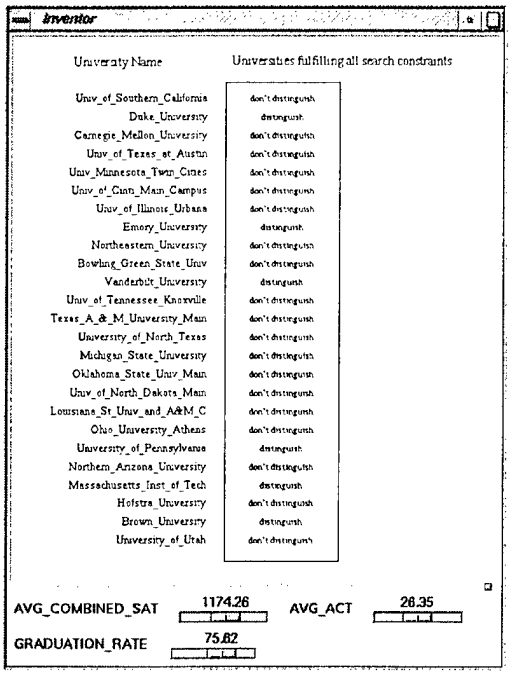


Figure E-2: Representing search results using text labels with "distinguish" or "don't_distinguish". Such results can be pre-attentively searched on through length matching.

For example in Figure E-1 the results of the search are shown with a simple "Y" or "N" symbol. In this case, even though technically the results are shown as labels, the labels used are so simple (a single letter) that it is similar to performing shape matching on glyphs which is a pre-attentive operation. Another possibility is to show the results using single words that have clearly different lengths (Figure E-2). In this way, users may also pre-attentively view the search results by filtering based on label length. As a result, this design can, with appropriate result phrases, be elevated to a much higher ranking. Currently however our automatic designer does not have this knowledge encoded within it. It is however not difficult for us to

make this addition in the future. Without this additional knowledge, the processing time for changing search conditions is also very significant. For each change, we must peruse through the entire data set to determine the general size of the search concepts and this operation costs an estimated 11750 msec. This time does not include the slider manipulation time which adds another 1790 msec to the operation.

E-1.8 Summary

Figure E-3 shows the estimated GOMS total time for all the designs analyzed in this section. The designs are ordered on the *x-axis* according to the costs assigned by our automatic system and the *y-axis* encodes the estimated total time taken to complete the given search task in msec. From Figure E-3 we see that there is an increasing trend from left to right. Designs that are higher ranked by our designer (to the left) also have lower estimated total times and those that are ranked lower (to the right) have higher estimated total times. Thus the GOMS estimations conform to the orderings assigned by our automatic designer.

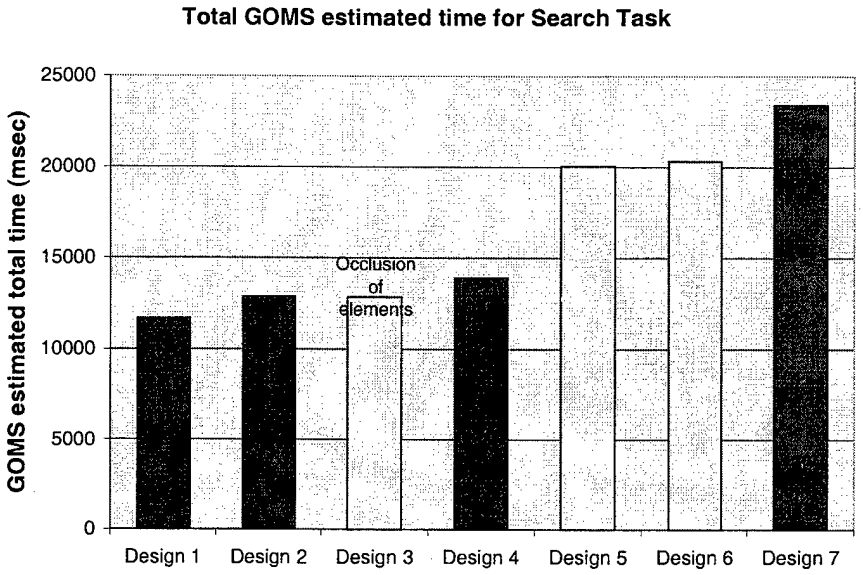


Figure E-3: GOMS estimated total time for Task 1 (search task). The designs are ordered based on increasing cost on the x-axis. The y-axis shows the GOMS estimated total time in msec. All pink bars indicate pure mapping designs (i.e. designs that can be generated with current state of the art automatic design research). All other bars are designs made possible by work outlined in this thesis.

The green bars in Figure E-3 are designs with data transforms techniques (i.e. they are designs that are made possible by work explored in this thesis) and the pink bars are designs that only contain mapping transform techniques (i.e. they are designs that can be generated with current state of the art automatic design technology). We see from Figure E-3 that a significant number of new designs are now possible that previously could not be generated (all of the green bars). More accurately, out of the twenty designs generated for this task, 4 were pure data transform designs, 6 were hybrid designs and 10 were pure

mapping designs. Thus 50% of the designs in the top twenty are new designs from the expanded data transform design space. We will see in the next two tasks analyzed that the percentage of new designs made possible by our work is even higher. This is not surprising since search tasks lend themselves very well to perceptual processing (i.e. there are many effective perceptual operators for performing search) but this is not true of computation tasks that are commonly difficult to perform perceptually. While the time gains of the expanded design space is less significant in this example compared to the next computation task, our examples show that the variety added in addition to the interactive components provide very interesting design alternatives to the user.

The contributions of our work can be further seen by comparing Design 1 which is the best design generated by our expanded designer and Design 3 which is the best design that can be generated by previous automatic systems. Design 1 computes the search results for the user based on the three constraints placed on *avg_combined_SAT, avg_ACT,* and *graduation rate*. Only universities that fulfill all three constraints are shown. The advantage of this display is that the filtering significantly reduces clutter and the number of data concepts (i.e. universities) and attributes that need to be shown. As a result, less display space is required for the visualization and there are no readability problems such as overly high object density or occlusion.

Design 3 on the other hand shows all of the data to the user (i.e. *avg_combined_SAT* scores on *saturation*, *avg_ACT* scores on *y-axis*, *graduation rate* on *x-axis* and *university name* on *labels*). Four data attributes are shown here compared to the one (*university name*) in Design 1. To find a university that has high values on the three search properties we need to look for marks that are to the upper right of the display with high saturation. While this is not a difficult perceptual operation, a large problem with this design is the occlusion that occurs among some of the elements which makes it impossible to read some of the relevant university names. The problem worsens for a larger data set. In addition, *saturation*, which is used in Design 3 to encode *avg_combined_SAT* score, cannot show the values with very high accuracy and as a result errors may occur (as we saw in our analysis with *University of Illinois*). In contrast the search results in Design 1 are highly accurate because they are pre-computed by the system. For these reasons, our automatic designer ranked Design 1 over Design 3.

Another interesting aspect of our analysis concerns the estimated added cost for changing the conditions of a search. In most part, the time required is approximately in line with the ranking of our designer as well. This is because the time taken for each condition change is dependent on a user's ability to quickly scan the visualization for a general change in size of the search object set. This operation is more difficult for the later (lower ranked) designs because the search results cannot be viewed pre-attentively unlike the earlier (higher ranked) designs. And it is exactly this pre-attentiveness that cause the designs to perform better in the first place and also score better in our design system. An exception is Design 3, which

as we noted can have negligible costs to changing search conditions because no input device manipulation is required and processing of the search results can be performed pre-attentively. Occlusion however is a problem and we discuss in appendix F how further additions to the automatic design process with graphical and rendering transforms may solve such readability problems and further expand the richness and quality of designs that can be automatically generated.

E-2 Task 2: Compute Task
Compute Total Non-salary Benefits Distributed by a Set of Universities

In this section we want to determine the results of a fairly complex computation. The task we chose is to determine the total benefits package given out by a set of universities to their faculty body. "Total benefits" refers to non-salary compensation. The size of the total benefits package given out by a university will indicate the general prosperity and quality of the university in terms of faculty size and general faculty incentives. The university data set used here was taken from *USNews*. This data set has three different faculty groups (*full*, *associate* and *assistant* professor) and two pieces of information for each group, namely the *total salary* and *total compensation* given out to each faculty type. To compute the total benefits given out by any particular university, we need to sum the differences between compensation and salary for each faculty group. The task specification entered into our automatic designer is shown below:

```
(compute   '(VALUE . ADD)
           (compute '(VALUE . SUBTRACT)
                       (lookup  '(OBJECT . nil)
                         '(VALUE . AVG_FULL_COMPENSATION))
                    (lookup '(OBJECT . nil) '(VALUE . AVG_FULL_SALARY))
                    )
           (compute '(VALUE . SUBTRACT)
                    (lookup '(OBJECT . nil)
                            '(VALUE . AVG_ASSOC_COMPENSATION))
                    (lookup '(OBJECT . nil)
                            '(VALUE . AVG_ASSOC_SALARY))
                    )
           (compute '(VALUE . SUBTRACT)
                    (lookup '(OBJECT . nil)
                            '(VALUE . AVG_ASST_COMPENSATION))
                    (lookup '(OBJECT . nil) '(VALUE . AVG_ASST_SALARY))
                    )
)
```

Task E-2: View the total benefits given to faculty for a set of universities.

From analyzing the data set using the visualization displays generated by our system, we were able to group the universities into four categories based on total benefits: 1) large, well-known state schools; 2)

relatively large second tier state schools & large private schools; 3) small, but very prestigious private schools; 4) smaller, less known state and private schools. The large, well-known state schools including *University of Austin Texas, University of Michigan,* and *University of Illinois* are the universities that have the largest total benefits package. This is because they have very large faculty bodies, in addition to providing good benefit incentives to their faculty members. The second group of universities are a mix between the large private schools (*Northeastern University, Southern University of California, MIT,* and *University of Pennsylvania*) and the mid-sized state schools (*Texas A&M University, University of Utah*). Universities in this group have relatively large faculty bodies and good faculty incentives. The third group of schools (*Brown, Duke, Emory, CMU*) are the very prestigious private universities that provide large incentives to their faculty members, but have a much smaller faculty size compared to the prior two groups. Finally, the last group of universities include the smaller private schools (*Hofstra University*) and state schools (*University of North Dakota Main,* and *Northern Arizona University*) with a small faculty and fewer benefits.

Since the task entered into our automatic designer is relatively general with respect to what can be done with the total benefit numbers, there are a set of different tasks that we could perform based on the output designs. For example we could try to categorize the universities based on total benefits as was done above, we could only find the top few universities giving out the most total benefits, we could compare the benefits from particular universities that we are interested in attending, etc. In the interest of simplicity, we chose a fairly small task for our GOMS evaluation:

Find the top four universities with the largest total benefits given to faculty.

As with the previous task we analyze a set of seven output designs, ordered from best (least cost) to worst (highest cost) based on the costs assigned to each design by our automatic system.

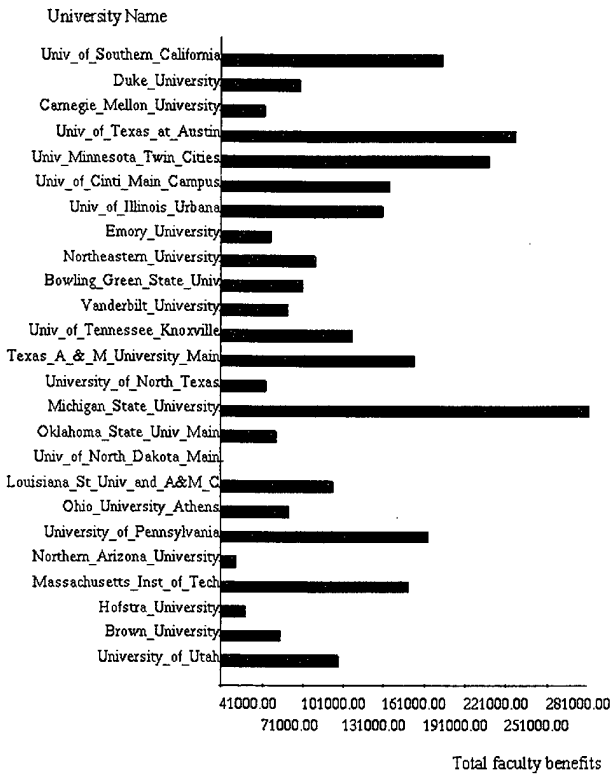
In all following GOMS procedures for this task we assume:

Baseline Assumptions for Task 2 (Computation Task):

1. Reading a university name is sufficient to commit it to memory until the end of the task, at which time the user is able to recall the top four university names read during the data analysis process. This may not be an accurate assumption because the ability of users to remember the university names depend on the length of the data analysis process. A longer analysis process would degrade short term recall and reduce the probability that users are able to remember the university names read. However, since we make this assumption for all designs, any recall failure can only have a negative bias towards the designs that already have a longer processing time (i.e. make processing time for a less efficient design even longer). This additional time therefore would not affect the initial time ordering/ranking of the evaluated designs.

E-2.1 Design 1

In this design, the entire *total benefits* computation is pre-processed with an internal data visualization technique. I.e. all the computation required is done internally by the system. The computed *total benefit* results for each university are then shown as *horizontal bar lengths* with the *university_names* on the *y-axis*.



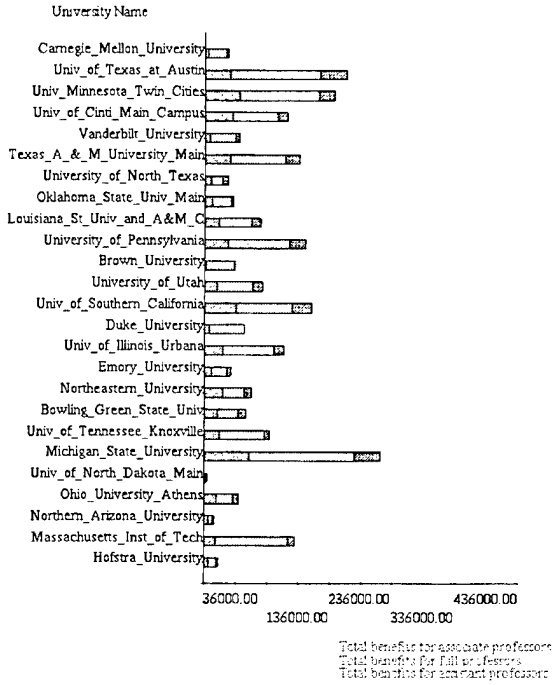
Using this design, we can easily pick the four universities with the highest *total benefits* package by choosing the four longest bars. This is a pre-attentive operation, which means that we do not need to attend to each bar to get the four longest. Instead we can simply scan from top to bottom and pick the bars that we see in order.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub-total (msec)	Total-time (msec)	Target objects (data or graphics)
Pick next long bar	Attend picking next long bar	50			bar 15, bar 4, bar 5, bar 1
	Initiate eye movement	50			
	Eye movement	30			
	Perceive next longest bar	100			
	Sub-total		230		

Scan to the left of chosen bar to read the university name. (Assume that on average a university name has three words, which is the general case in our data set)	Attend get university name	50			1. Michigan State University, 2. University of Texas at Austin, 3. University of Minnesota Twin Cities, 4. University of Southern California
	Initiate eye movement	50			
	Eye movement	30			
	Read university name (3 word name) = 3 * 290	870			
	Verify name	50			
	Sub-total		1050		
Total time to process each bar	1050 + 230	1280			
Repeat *4* times for each of the top universities	4 * 1280			**5120**	
Total time				**5120**	

This design is very uncluttered and simple with most of the data summarized. We can perform our task very quickly and with high accuracy.

E-2.2 Design 2

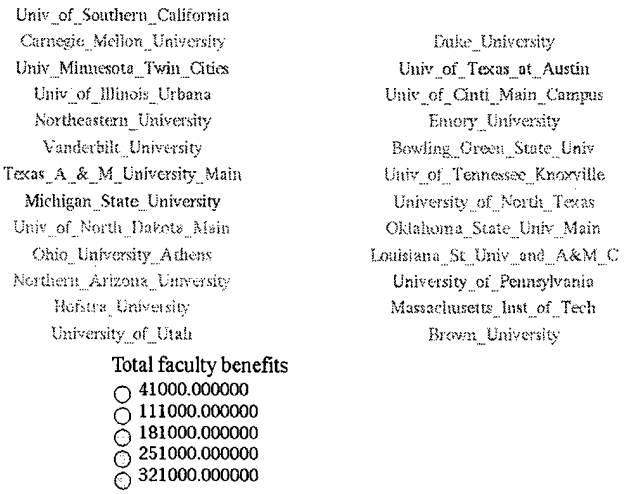


This visualization is a hybrid design. Part of the total benefits computation (i.e. the *subtraction* of *total_salary* from *total_compensation* for each faculty type) is performed internally, and the other part (i.e. the *summation* of *total_benefits* for each faculty type) is left to the user to perform perceptually.

Specifically, the *total_benefits* for each faculty type are shown using the *x-lengths* of three differently colored bars. The green bars represent *total_benefits* for *full professors*, the red bars represent *total_benefits* for *associate professors*, and the purple bars represent *total_benefits* for *assistant professors*. In addition, the bars are stacked to facilitate perceptual computation of the *summation* task. In this design, we can use the same procedure as Design 1. There is more perceptual complexity here compared to Design 1 (i.e. a separate bar is used to represent the *total_benefits* for each faculty type), but the stacking technique is so effective at grouping the objects and helping users perceptually perform the *summation* task, that the additional perceptual load for this design is negligible. For these reasons, our automatic system ranked this design second best. We would like to point out, however, that because of the greater design complexity compared to Design 1, it would initially take the user a longer time to understand this graphic. However, as was noted in our base assumptions, we do not take initial graphic understanding costs into account but instead assume that the user is an expert and already has great familiarity with the visualization designs generated. On the other hand, once understood this design includes more details that may be useful for subsequent tasks.

E-2.3 Design 3

In this design, the entire *total benefits* computation is pre-processed as in Design 1. However unlike Design 1, we map the results to *label saturation* instead of to *x-length* (position) as was done previously. The *labels* themselves show the *university_names*. *Saturation* while pre-attentive like *position*, is significantly less accurate for making quantitative value judgements. This is because our eye can only differentiate relatively **large** differences in *saturation* values. Because of the lower encoding accuracy, this design is ordered below both Design 1 and Design 2.



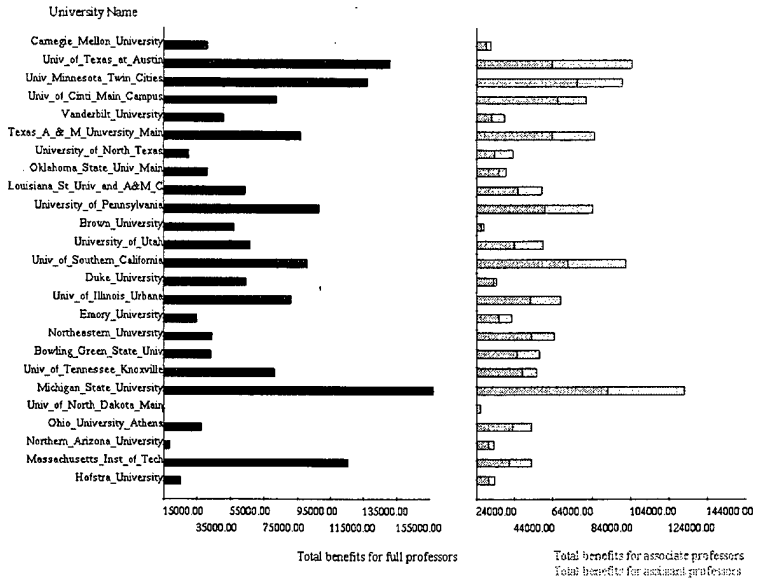
Using this design, we can find the four universities with the highest *total benefits* package by going to the four most highly saturated university names. This is a pre-attentive operation, thus we do not need to attend to each label. However, in the design above, we found ***seven*** universities with high *saturation* values and it is difficult to pick the four most saturated out of this set of seven because of the difficulty in making *saturation* judgements. As a result the GOMS estimated time is higher, because of the additional three university concepts that must be processed. In addition, the error rate is also higher, however that is not taken into account in the GOMS estimation.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub-total (msec)	Total-time (msec)	Target objects (data or graphics)
Scan and pick next highly saturated label	Attend picking next saturated label	50			Label 1, 3, 7, 8, 15, 23, 24
	Initiate eye movement	50			
	Eye movement	30			
	Perceive saturated label	100			
	Sub-total		230		
Read university name	Attend read university name	50			1. University of Southern California, 2. University of Minnesota Twin Cities, 3. Texas A&M University Main, 4. Michigan State University, 5. University of Texas at Austin, 6. University of Pennsylvania, 7. Massachusetts Institute of Technology
	Read university name (3 word name)	870			
	Verify name	50			
	Sub-total		970		
Time taken to process each saturated label	(230 + 970)	1200			
Repeat process 7 times for each saturated label	7 * 1200			**8400**	
Total time				**8400**	

In the design above, there is an error rate of 43% (3/7). This error rate will increase the closer the *total benefit* numbers are to each other. As the error rate increases, so does the total time taken because there are more universities that need to be processed than necessary for the task. The one advantage of this design over the two previous examples is that no eye movement is necessary to get from the *total benefits* object to the university name because both *university_name* and *total_benefit* is encoded within the same *label* glyph.

E-2.4 Design 4

This design is similar to Design 2 in that it is a hybrid design with the *total_benefits* for each faculty type computed internally, and with the *summation* task left to be performed perceptually by the user. Unlike Design 2 however, this design splits the information over two chart spaces, the left space shows the *total_benefits* for *full professors* on the *x-lengths* of the bars and the right chart shows the *total_benefits* for the other two faculty categories. Because there is less integration in the visualization (i.e. two spaces are used instead of just one as in all the previous designs) our designer recognizes that it would be more difficult to perform the *summation* task and as a result this design is given a higher cost (assigned a lower ordering).



To find the four universities with the highest *total benefits* package with this design, we need to scan across the two chart spaces and combine results from both charts. An important issue in determining the GOMS procedure here is the consistency of the axis scales in the two charts. Our system by default scales the axes of the charts according to the *min* and *max* values within each chart. This is so that the encoded values can be determined with the greatest amount of accuracy given the available space. If we were to scale all of the charts to a single consistent scale, some of the bar sets could get significantly dwarfed, thereby making it difficult for users to lookup their values if needed. However for the computation task here it is advantageous to make all of the *x-axis* intervals consistent because that can significantly facilitate the comparison of bar lengths across charts and this will allow for more effective task performance. Thus we start our GOMS procedure for this design by having the user determine the axes intervals in each chart and then re-scaling the axes through interactive operations if the intervals are not identical. In this design, the axis intervals are identical to begin with at 20k, thus no additional re-scaling is required.

Once we have scaled all charts to consistent axis intervals, we start processing the left-most chart based on the assumption that in Western convention, reading is from left to right, top to bottom. In the first chart we attend to, we scan for all the longer bars. Note that in this case, we may need to scan for more than just the four longest bars depending on the corresponding bar lengths on the related chart to the right. In this design we found 9 candidate universities with relatively long bars in the first chart. Out of these nine candidate bars, three are clear winners, *Michigan State University, University of Texas at Austin,* and *University of Minnesota Twin Cities*, all have the top longest bars in both charts. The fourth position however is less clear with several close possibilities including *University of Cincinnati Main Campus, Texas A&M university Main, University of Pennsylvania,* and *University of Southern California*. There are several ways in which we can process these four candidates. We describe two alternative methods here:

Case 1: Comparison of bar length differences

The most effective way, is to get the difference in bar lengths between pairs of candidate bars in both charts and then pick the university with the larger difference. For example, initially, we get the difference in length between the *Texas A&M University Main* bar with the *University of Cincinnati Main Campus* bar on the left chart. We then repeat this for the chart to the right. In this case, both *Texas A&M University Main* bars exceed the *University of Cincinnati Main Campus* bars in length, thus the *Texas A&M University Main* concept is chosen as the current fourth possibility. When processing the *University of Pennsylvania* concept however we find that the *University of Pennsylvania* bar is greater than the *Texas A&M University Main* bar in the left chart, but less in the right chart. In this case, we actually need to compare the two length differences and determine which is larger. For this design and data set the difference is larger in the left chart, thus *University of Pennsylvania* replaces *Texas A&M university Main* as the fourth choice. We then repeat this process for *University of Pennsylvania* and *University of Southern California*. Note that for this method to work, we first need to ensure that the axis intervals in both charts are identical, which we did at the start of the GOMS procedure.

Case 2: Comparison of total_benefit values

Another alternative method for processing the four candidate bars is to read-off their total benefit values from the *x-axis*, mentally combine the benefit values for both charts, and then compare these combined values, picking the highest one. Note that translating an *x-position* into a *total_benefit* value is a somewhat complex operation. This is because there are only labeled tick marks on the *x-axis* every so often and if an *x-position* falls in between two tick marks, we must estimate where it falls within the tick-mark interval before we can convert the *x-position* into a *total_benefit* value. In the GOMS sequence below, we estimate inter-interval positions by dividing up an interval into quarters and then estimating which quarter the *x-position* falls into. This method should be effective in most cases unless dwarfing occurs on the graphical property we are attempting to translate (refer to appendix F on details on dwarfing).

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Compute size of tick-interval on the left chart	Attend scan first x-axis label	50			20k interval
	Initiate eye movement	50			
	Eye movement	30			
	Read number	290			
	Verify number	50			
	Attend scan second x-axis label	50			
	Initiate movement	50			
	Eye movement	30			
	Read number	290			
	Verify number	50			
	Mental subtract number 1 from number 2 (assume 2 significant figures)	250			
	Sub total		1190	**1190**	
Compute size of each tick on the right chart.		1190		**1190**	20k interval
Since both charts have consistent axis intervals (20k) no additional load is required for re-scaling.					
Pick next long bar in left-most chart.	Attend picking next long bar	50			1. Michigan State University, 2. University of Texas at Austin, 3. University of Minnesota Twin Cities, 4. Massachusetts Institute of Technology, 5. University of Pennsylvania, 6. Texas A&M University Main, 7. University of Southern California, 8. University of Cincinnati Main Campus, 9. University of Illinois
	Initiate eye movement	50			
	Eye movement	30			
	Perceive longest bar	100			
	Sub-total		230		
Scan to same row on chart to the right	Attend scan to right chart	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive bar	100			
	Sub-total		230		
Determine whether current bar length is long wrt. other bars in chart. (note that comparison here is pre-attentive)	Attend compare bar length	50			
	Compare	50			
	Verify result	50			
	Sub-total		150		
If bars are long, store bar position in STM as a potential candidate.					
Time taken to process each bar pair	230 + 230 + 150	610			
Repeat for each long bar on first chart	9 * 610			**5490**	9 bars on left chart which are potential candidates.

There are then 4 bars that seem to have approximately the same combined lengths.

1. University of Cincinnati Main Campus,
2. Texas A&M university Main,
3. University of Pennsylvania,
4. University of Southern California

In the next steps we ascertain which of these candidates has the largest total benefit value by comparing bar length differences (*case 1*) or converting the bar lengths into *total benefit values* and then comparing those values.

CASE 1: *Comparing bar length differences*

Scan to first unclear bar	Attend scan to right chart	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive right edge of bar	100			
	Sub-total		230		
Compare current bar length with previous candidate bar length. Assume that the user can then store this length in STM.	Attend compare	50			1. Texas A&M University Main with University of Cincinnati Main Campus, 2. Texas A&M University Main with University of Pennsylvania, 3. University of Pennsylvania with University of Southern California
	Attend process previous candidate bar	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive right edge of bar	100			
	Compare	50			
	Verify results	50			
	Sub-total		380		
Repeat this process in the chart to the right					
Compare the two length differences when the longer bar concept is different for each of the two charts	Attend compare	50			1. Texas A&M University Main with University of Pennsylvania, 2. University of Pennsylvania with University of Southern California
	Compare	50			
	Verify results	50			
	Sub-total		150		

Total time taken for CASE 1 includes:

1. Time taken to scan for the lengths of all the candidate bars. There are 4 candidates, and two bars per candidate thus total time = 4 * 2 * 230 msec. = 1840 msec.
2. Time taken to compare the current candidate bar with the previous candidate bar. We need to perform this comparison 3 times because there are four candidates (for the first candidate no comparison is required). Since there are two charts, the total time here = 3 * 2 * 380 msec. = 2280 msec.
3. Time taken to compare the two length differences. This operation has to be performed twice for the two cases where the longer bar concept is different in each of the two charts. Total time = 2 * 150 msec. = 300 msec.

Total time for CASE 1	1190 + 1190 + 5490 + 1840 + 2280 + 300			**12290**	

CASE 2: *Comparing the converted total benefit values*					
Scan to first unclear bar	Attend scan to right chart	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive right edge of bar	100			
	Sub-total		230		
Scan to axis to read bar length	Attend determine benefit	50			
	Initiate eye movement	50			
	Eye movement down (assume there is no difficulty with length from actual axis)	30			
	Sub-total		130		
Estimate tick interval position of bar (i.e. where within a tick interval the end point of the bar is situated)	Note bar length pos. on current tick interval	100			
	Determine in quarters the position of the bar length within the current interval tick.	50			
	Sub-total		150		
Convert distance to benefit based on length of 1 axis step	Convert to total benefit value (** refer to division by halving description below). Conversion can be either on: no interval, quarter interval, half interval, or three-quarter internal.	375			
Add with last interval tick	Mental add (2 significant numbers)	250			
Total time taken to lookup the total benefits of the first bar from the axis.	(230 + 130 + 150 + 375 + 250)	1135			
Total time taken to lookup the total benefits of the bar on the right chart		1135			
Add benefits from both charts	Mental add (We assume that the numbers have 2 significant figures based on the axis labels in the design)	250			Texas A&M university Main = approx 100k + 89k = 189k University of Pennsylvania = approx 110k + 89k = 199k; University of Southern California = approx. 105k + 104k = 209k
Total time taken to process each unclear row	Subtotal for unclear bars (1135 + 1135 + 250)	2520			
Repeat for each unclear bar pair	4 * 2520	10080			
Total time for CASE 2	1190 + 1190 + 5490 + 10080			**17950**	

Not surprisingly *case 1* results in a much lower total estimated time than *case 2*. This is because in *case 1*, no mental computations are necessary, nor do we need to convert from *bar length* to *total benefit* values. However, we should point out that in cases where the bar length differences in both charts are very close, it may be difficult to perform accurate comparisons across different charts (i.e. it may not be possible to remember the bar length difference in the first chart with enough accuracy that we can correctly compare it in the second chart.). In this design the bar length differences are large enough that this does not become a problem. If such a problem does arise however, we may need to resort to the *case 2* method.

Note that the further processing required for the four universities in the GOMS algorithm above (as represented by the *case 1*or *case 2* methods) is a result of the lack of integration in the graphic. The additional steps taken show some of the problems associated with dividing up related computation data into multiple different spaces. Interestingly enough whichever method is used does not alter the time ranking of this design. All subsequent designs have higher estimated total time compared to both *case 1* and *case 2* estimation of Design 4.

Mental Add and Subtract

In the above GOMS sequence, there were several mental addition and subtraction operations. Each of these operations are performed on numbers with 2 significant figures. The detailed breakdown of these operations are as follows:

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Mental add or subtract	Attend computation	50			
	Add first digit	50			
	Carryover	50			
	Add second digit	50			
	Verify results	50			
	Sub-total		250		

Note that to perform mental computations on numbers with more significant figures, we just add in 100 msec. for each significant figure. Essentially the time taken for a mental add or subtract is 150 + (n-1) * 100 where n is the number of significant digits in the numbers to be computed.

Mental Division

In the GOMS sequence above there is also a mental division operation. Specifically, it occurs when we need to convert a position that falls in between two tick marks back into its *total_benefit* value. Since in this case we only ever divide inter tick mark positions into quarters (i.e. quarter of a tick, half of a tick, or three quarters of a tick), we may simplify the mental division operation by simply halving the interval value as is shown below:

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Mental division by halving	Attend computation	50			
	Halve first digit	50			
	Halve second digit	50			
	Add both numbers	50			
	Verify results	50			
	Sub-total		250		

This allows us to compute the required division when the *x-position* is exactly in the middle of a tick mark. To perform the other cases, we can combine both mental subtraction and "division by halving" as is shown below. The table below lists all the inter-interval cases and the total times required. Since each of the cases is equally likely, we compute the time of any one case by taking the average time of all the cases, i.e. (0 + 250 + 500 + 750) / 4 = 375.

Possible cases	How to convert to total benefit value	Total time (msec)
No interval	No operation	0
Half interval	division by halving	250
Quarter interval	perform division by halving twice	500
Three quarter interval	Perform quarter interval and subtract that from a tick interval	500 + 250

E-2.5 Design 5

This design is a pure internal design, where the computation has been processed internally and only the final results are shown. Unlike previous cases however, where *position* (Design 1) and *saturation* (Design 3) are used, this design uses *labels* to show the total benefit values. Labels are not very effective for this task because they cannot be searched on pre-attentively. As a result, we must attend to each label individually and mentally compare the numbers with each other. This is a cognitively intensive operation and as a result this design is ranked much lower by our automatic designer. In addition, there is also significant state that the user has to keep track of and it is difficult to keep them all in STM (it is difficult to remember numbers). As a result, in our GOMS procedure, the user tracks state information by using their fingers as pointers.

University Name	Total faculty benefits
Univ_of_Southern_California	209920.00
Duke_University	101040.00
Carnegie_Mellon_University	74240.00
Univ_of_Texas_at_Austin	265209.00
Univ_Minnesota_Twin_Cities	245474.00
Univ_of_Cinti_Main_Campus	169492.00
Univ_of_Illinois_Urbana	164436.00
Emory_University	79007.00
Northeastern_University	112416.00
Bowling_Green_State_Univ	103558.00
Vanderbilt_University	91711.00
Univ_of_Tennessee_Knoxville	140956.00
Texas_A_&_M_University_Main	188679.00
University_of_North_Texas	75046.00
Michigan_State_University	320788.00
Oklahoma_State_Univ_Main	82513.00
Univ_of_North_Dakota_Main	40730.00
Louisiana_St_Univ_and_A&M_C	126133.00
Ohio_University_Athens	92852.00
University_of_Pennsylvania	199225.00
Northern_Arizona_University	52341.00
Massachusetts_Inst_of_Tech	183356.00
Hofstra_University	59244.00
Brown_University	86175.00
University_of_Utah	129774.00

In our GOMS procedure the user utilizes four fingers to keep track of the current four highest total benefit numbers. At the start of the procedure, the user picks the first four numbers as the current four highest, thus fingers are placed next to each of these numbers. In addition, the user also determines which finger points to the lowest of these four numbers (i.e. the *lowest max*) and generally what its total benefit value is (e.g. approx 74k). Subsequently, the user processes the following numbers, comparing them to the *lowest max* value. When a total benefit figure is found that is higher than the current *lowest max*, the user updates the current state by changing a finger position from the current *lowest max* position to the new number position. Note that we assume this only requires a finger move. It may sometimes, however, be necessary to move several fingers on these updates to keep our fingers from getting tangled up. However in such cases, the different finger moves would usually happen in tandem (simultaneously) thus the total time for the operation will be approximately equal to that of a single finger move. In addition to changing finger positions, we must also compute a new *lowest max* figure and keep its position and value in short term memory. We continue this process until we have processed all numbers in the table. At the end of the process our four fingers should be pointing to the four universities with the four highest total benefit values.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Place fingers on first four numbers indicating that they are currently the four largest numbers.	Attend point to first four numbers	50			209920, 101040, 74240, 265209
	Initiate hand move	50			
	Move hand from lap to first four entries on screen	350			
	Sub-total		450	**450**	
Scan next number	Attend read next number	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read number	290			
	Verify number	50			
	Sub-total		470		
Compare if number is more than *lowest max* pointed to by fingers	Attend number > current *lowest max*	50			
	Compare number with *lowest max*	50			
	Verify result	50			
	Sub-total		150		
Minimum total time taken to attend to each number	470 + 150	620			
If number is greater than *lowest max*, point finger to current number and remove finger from the previous *lowest max.*	Attend change finger positions	50			245474 → 74240 (max = 1 step) 169492→101040 (max = 2 steps) 188679 → 169492 (max = 7 steps) 320788 → 188679 (max = 2 steps) *Total move = 12 steps* Average move = 12/4 = 3 steps
	Initiate finger lift	50			
	Lift finger	60			
	Initiate finger move	50			
	Move finger $100 * (\log_2 (1.5/0.5)+.5)$	181			
	Initiate finger drop	50			
	Drop finger	60			
			501		

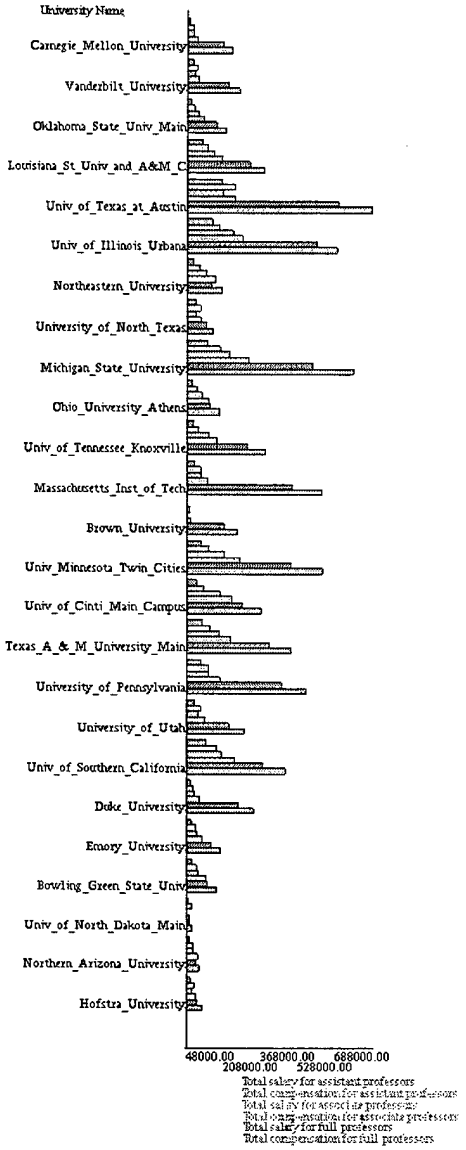
Time taken to move finger in the calculation above we estimate as follows:
We assume that the visualization is enlarged so that each label entry is at least the width of a finger. Or else, finger pointing would be very difficult. Thus,
Height of widest finger = Height of each entry = 0.5 inches
For this particular design and data set, we calculated 4 necessary finger moves, with an average total distance move of 3 steps = 3 * 0.5 inches = 1.5 inches
Thus using Fitts Law, the estimated time for movement is $100 * (\log_2 (1.5/0.5)+.5) = 181$ msec

In the general case, we can assume that any single move will be less than ¼ of the total distance because we are looking for four of the largest numbers, and on average each of these numbers should be spread out one in each quartile. Since there are 25 elements in the data set, each move should on average be 24/4 = 6.25 steps = 3.125 inches
Thus using Fitts Law, the estimated time for movement is $100 * (\log_2 (3.125/0.5)+.5) = 275$ msec

Compare numbers pointed to by four fingers to get the new *lowest max* value and store in STM.					
Compare a single pair of numbers	Compare result on finger 1 and finger 2	50			
	Attend read number on finger 1	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read number 1	290			
	Verify number 1	50			
	Attend read number on finger 2	50			
	Initiate eye movement	50			
	Eye movement	30			
	Read number 2	290			
	Verify number 2	50			
	Attend compare	50			
	Compare #-1 and #-2	50			
	Verify result	50			
	Sub-total		1140		
Up to three pairs of numbers need to be compared to get the *lowest max* value. However note that only two numbers need to be read on the first comparison, on subsequent comparisons, only one number needs to be read because the other number is carried over from the previous computation.	1140 + (2 * 670)	2480			
Total time taken include time to process each number and the additional time of updating and computing new *lowest max* figures as necessary.					
Process all label entries in table. There are 25 numbers -4 of the first numbers = 21 numbers	21 * 620	13020		**13020**	
We need to perform four *lowest max* replacements in this particular visualization	4 * (501 + 2480)	11924		**11924**	
Total time	450 + 13020 + 11924			**25394**	

Note that this procedure requires significant cognitive and STM loads. The additional fatigue caused by these operations (compared to perceptual operations) are not taken into account in the GOMS procedure. This additional fatigue can potentially reduce the length of time a user is able to perform tasks effectively and with low error rates.

E-2.6 Design 6



This design uses only mapping techniques, i.e. all of the task data are mapped to graphics and it is up to the user to perform the entire computation task perceptually. There are six bars shown per university. Each pair of bars represent the *total_compensation* and *total_salary* for each of the three faculty types (*full_professor, associate_professor,* and *assistant_professor*). Under normal circumstances, this graphic would be very difficult to use because the computation involved in this task is very complex. Specifically, users would need to perceptually determine the difference in bar lengths between each pair of bars (of which there are 3 per university) and then sum up the bar length differences for each of the three bar pairs. This operation would require significant finger pointing etc, if not help from using measurement tools (i.e. a ruler). Once computed, the total benefit lengths would need to be marked on the paper so that at the end

of all the processing, the user can compare all the perceptually computed lengths together. Because of the high computation load and the additional perceptual clutter from having to show six bars, the design is given a significantly high cost by our automatic design system.

Although normally such a design would be close to being unusable, in this example we are able to exploit patterns in the data distribution and realize significant perceptual shortcuts. Specifically, from looking at the bars, it becomes apparent that a significant portion of total benefits given out by a university goes to full professors (represented by the two purple bars). I.e. the difference in length between the two purple bars are comparably much higher than the difference in lengths of the other bar pairs. In addition, it is the universities with medium to long bars (greater total salaries and compensation) also have greater pair-wise bar length differences. In summary to perform our task we only need to process the universities with longer bars and for each of those, we only need to get the difference in lengths between the two purple bars. Since there are more than four possible candidate long bar sets, we must compare each perceptually determined bar length difference with all other previous bar length differences to choose the purple bars with the greatest length disparity. In this design, it is difficult to pre-attentively determine the cases where the length differences between the two purple bars are largest. Thus we process all university concepts with medium to long bars, where there are noticeable differences between the two purple bars (i.e. we need to process more university bars then in the previous designs).

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Scan to first set of longer purple bars	Attend scan	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive bar set	100			
	Sub-total		230		
Get length difference between *full_professor total_compensation* bar and *full_professor total_salary* bar	Attend get length difference	50			
	Perceive difference	100			
	Verify results	50			
	Sub-total		200		
Compare bar length difference with all previous bar length differences and discard the one with the least difference. Here we assume that the user can remember the positions of the current set of four universities with the greatest bar length differences. We make this assumption here, and not in the previous case because here, the user can use the bar lengths as a general tag for the relevant bar objects. And since length searching is pre-attentive, it would not be difficult to get to the desired bars.					
Time taken to make a single comparison between the current bar length difference with one of the current largest bar length differences	Attend compare with current largest bar length difference	50			
	Attend scan to one of the largest difference bars	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive difference	100			
	Attend compare	50			
	Compare	50			
	Verify results	50			
	Sub-total		430		

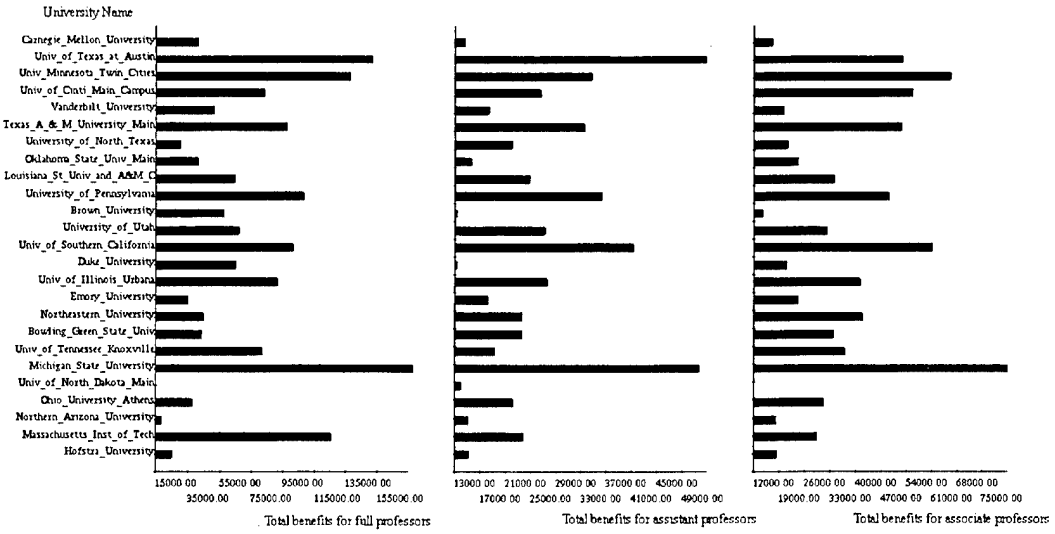
There are 4 comparisons that need to be performed	4 * 430		1720		
Total time for this task includes processing each of the long bar sets, of which there are 13, and performing the comparison with the current 4 largest differences, of which we need to perform (13 – 4) = 9 times. The 13 bars processed include: 1. Louisiana State University and A&M C. 2. University of Texas at Austin, 3. University of Illinois, 4. Michigan State University, 5. University of Tennessee Knoxville, 6. Massachusetts Institute of Technology, 7. University of Minnesota Twin Cities, 8. University of Cincinnati Main Campus, 9. Texas A&M University Main, 10. University of Pennsylvania, 11. University of Utah, 12. University of Southern California, 13. Duke University					
Repeat 13 times for 13 of the tallest bars	13 * (230 + 200)			5590	
Repeat 9 times	9 * 1720			15480	
Total time	5590 + 15480			**21070**	1. University of Texas at Austin, 2. Michigan State University, 3. Massachusetts Institute of Technology, 4. University of Minnesota Twin Cities

It is important to note that in the GOMS procedure above we **do not** take into account the time taken to notice the perceptual patterns in the display nor the time needed to come up with a plan on how to exploit those patterns. We should also recognize that because we are taking significant perceptual shortcuts, we may derive erroneous results in some cases. For example in the design above *Massachusetts Institute of Technology* is chosen instead of *University of Southern California* as one of the universities with the greatest total benefits even though that is incorrect. This is because *Massachusetts Institute of Technology* has very large total benefits given to full professors and very small benefits for all other groups whereas in *University of Southern California* the total benefits are more evenly distributed to all three faculty groups. Because we are only using total benefits for full professors as a yardstick for total benefits for all faculty, errors similar to this will arise for data concepts that do not fit the general perceptual pattern we used to reduce task complexity. Finally, the perceptual shortcuts used in the GOMS procedure here may not be utilized in all cases. We were able to exploit them here only because of the particular data distribution we are examining.

Our automatic designer placed this design below Design 5 even though the GOMS procedures show that this design is more time efficient than Design 5 because our system cannot take into account the specialized perceptual shortcuts that may be exploited in this particular case. Automatically identifying such patterns and taking them into account in the automatic design process, is a difficult but certainly interesting problem that is left for future work.

E-2.7 Design 7

This design is a hybrid data + mapping visualization. The total benefits for each faculty is computed internally, and their results are shown in three separate charts. Each chart represents the total benefit numbers for each faculty group. The GOMS procedure used here is similar to Design 4, except that in Design 4 the information is separated over two charts instead of three. Because the information is not well integrated, the design gets a higher cost compared to all previous visualizations.



The GOMS procedure used for this design is similar to the one used in Design 4, the only difference being that there is an additional chart space that we must process. As was the case previously, we begin by determining the axis intervals for the three charts. Unlike Design 4, the axis ranges for the three charts are different. The left-most chart has an axis interval of 20k, the middle chart has an axis interval of 4k and the right-most chart has an axis interval of 7k. As a result the user must re-scale the *x-axes* of the three charts to equalize their axis intervals. The most efficient and direct interactive method to achieve this would be for the user to type in the desired min-max values in each chart. First of all, the user must determine the min and max values to use. This is achieved by comparing and min and max *x-axis* values in each of the three charts. Once the proper values are determined, users need to select the two min and two max axis values

Total time to get min value	(470 * 3) + (200 * 2)	1810			
Total time to get max value		1810			
Determine min-max values to use	Attend get min-max-values	50			
	Get min value	1810			12000
	Get max value	1810			155000
	Sub-total		3670	**3670**	
Now we determine the total time for interactively re-scaling the axes.					
Click on first min or max value to change	Attend click	50			
	Initiate mouse move	50			
	Mouse move	100			
	Initiate click	50			
	Upstroke	60			
	Down-stroke	60			
	Sub-total		370		
Type in min/max value	Attend input in the min value	50			
	Initiate move hand to numeric keypad	50			
	Move from mouse to numeric keypad	132			
	Attend type value	50			
	Type in value: Number with move required takes 40 + 60 + 60 = 160 msec. (move, upstroke, downstroke) Number with no move required takes 60 + 60 = 120 msec. (upstroke, downstroke) Value 12 000 takes (3 * 160) + (2 * 120) = 720 msec. Value 155 000 takes (3 * 160) + (3 * 120) = 840 msec.	720/ 840			
	Verify results	50			
	Sub-total for min entry		1082		
	Sub-total for max entry		1202		
Move hand back to mouse	Attend move hand to mouse	50			
	Move from numeric keypad to mouse	132			
	Sub-total		182		
Total time to actually enter in the proper min-max values includes: 1. Time taken to enter in the min values = 2 * (370 + 1082 + 182) = 3268 msec. 2. Time taken to enter in the max values = 2 * (370 + 1202 + 182) = 3508 msec. Total time for entry = 6776 msec.					

that must be changed and enter in the newly computed min-max values so that all chart ranges are identical. We assume that the numeric keypad is used for the number entries so that once the user's hand is over the keypad no further hand movements are necessary. Between each number entry however users must move their hand between the keypad and the mouse because before an entry can be made the user must first choose which axis number they are altering by clicking on it.

Once the axis intervals are all consistent, we start processing the left-most chart and find all long bars within it. For each long bar we scan to the right and determine whether the respective bars in the other charts are relatively long as well. Many of the operations in the GOMS sequence below utilize operation times that have already been computed in Design 4. As was also the case in Design 4, three universities stand out as having very high total benefit values (*Michigan State University, University of Texas Austin, University of Minnesota Twin Cities*), however, there are four candidate universities (i.e. four rows) that could fill the fourth position. As in Design 4, we outline the two alternate methods here for determining which of these four candidate rows has the highest total benefit value.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Compute size of tick interval on the left-most chart.	Total time taken from Design 4	1190		**1190**	20k interval
Compute size of tick interval on the middle chart.		1190		**1190**	4k interval
Compute size of tick interval on the right-most chart.		1190		**1190**	7k interval
Re-scale all axes so that they have similar ranges:					
Read single min value	Attend get min value	50			15000
	Initiate eye movement	50			
	Eye movement	30			
	Read number	290			
	Verify number	50			
	Sub-total		470		
Read next min value		470			min(15000, 13000)
Compare two mins and pick the smaller one. Since the numbers only have 2 or 3 significant figures we assume a simple mental comparison is sufficient here.	Attend compare	50			13000
	Compare numbers	50			
	Compare second digit	50			
	Verify results	50			
	Sub-total		200		
Read next min value		470			12000
Compare min values		200			min(12000, 13000)

Now that we have re-scaled the three charts, we proceed to processing the bars.

Pick next long/medium bar on leftmost chart	Attend picking next long bar	50			1. Michigan State University, 2. University of Texas Austin, 3. University of Minnesota Twin Cities, 4. MIT, 5. University of Pennsylvania, 6. Texas A&M University Main, 7. University of Southern California, 8. University of Illinois, 9. University of Cincinnati Main
	Initiate eye movement	50			
	Eye movement	30			
	Perceive next long bar	100			
	Sub-total		230		
Scan to bar on same row to the right chart	Attend scan	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive bar	100			
	Sub-total		230		
Compare bar length with other bars in chart to see if bar has comparatively long length	Attend compare	50			
	Compare	50			
	Verify results	50			
	Sub-total		150		
Scan to bar on the same row to the last chart and perform similar comparisons	230 + 150	380			
Total time for processing each row of bars	(230 + 230 + 150 +380)	990			
Repeat 9 times for 9 longest bars in leftmost chart	(9 * 1090)	8910		**8910**	

There are then 4 bars that seem to have approximately the same combined lengths.

5. University of Cincinnati Main Campus,
6. Texas A&M university Main,
7. University of Pennsylvania,
8. University of Southern California

In the next steps we ascertain which of these candidates has the largest total benefit value by comparing bar length differences (*case 1*) or converting the bar lengths into *total benefit values* and then comparing those values. Note that the following time estimates are all taken from Design 4 and adapted to suit this design which has 3 charts instead of 2.

Total time taken for CASE 1 includes:

1. Time taken to scan for the lengths of all the candidate bars. There are 4 candidates, and three bars per candidate thus total time = 4 * 3 * 230 msec. = 2760 msec.
2. Time taken to compare the current candidate bar with the previous candidate bar. We need to perform this comparison 3 times because there are four candidates (for the first candidate no comparison is required). Since there are three charts, the total time here = 3 * 3 * 380 msec. = 3420 msec.
3. Time taken to compare the three length differences across three charts. This operation has to be performed for all three comparison pairs and two comparisons are needed per pair. Total time = 2 * 3 * 150 msec. = 900 msec.

Total time for task assuming CASE 1 = Time to determine axes intervals + Time to determine min-max values + Time to re-scale axes + Time to process all 9 candidate elements + Time to perform CASE 1 processing = (3 * 1190) + 3670 + 6776 + 8910 + (2760 + 3420 + 900) = **30006 msec**
Total time taken for CASE 2 includes: 1. Time taken to convert the lengths to benefit values and then add those values = left chart conversion (1135) + middle chart conversion and addition (1135 + 250) + right chart conversion and addition (1135 + 250) = 3905 msec. We have to repeat this for each unclear bar, thus total time = 4 * 3905 = 15620 msec.
Total time for task assuming CASE 2 = Time to determine axes intervals + Time to determine min-max values + Time to re-scale axes + Time to process all 9 candidate elements + Time to perform CASE 2 processing = (3 * 1190) + 3670 + 6776 + 8910 + 15620 = **38546 msec**

We want to mention that *case 1* computation for this design can be difficult because unlike Design 4 there are three charts here, thus users must compare length differences between pairs of bars across three charts, and it may be difficult to accurately maintain the bar length differences across this many graphical regions. It is also interesting to note that the time taken to re-scale the axes is fairly large, taking about 10 seconds (10000 msec), and as a result the percentage difference in time between *case 1* and *case 2* is less pronounced here compared to Design 4.

E-2.8 Summary

As can be seen in Figure E-4 the estimated GOMS time for the various designs analyzed for this computation task conform to the orderings assigned by our automatic designer. Also note that a significant number of new designs are now possible that previously could not be generated (all of the green bars). In this compute task there are far fewer mapping designs that make it to the top design spaces compared to the search task. Given our design space limit size of 15000 nodes, we generated 15 designs for this task. Out of these 15 designs 4 were purely data designs, 10 were hybrid data + mapping designs, and only 1 was a pure mapping design. In addition as can be seen from the chart below, many of the most effective designs all contain data transformation techniques (i.e. they are designs that could not previously be generated). This is not surprising, since computation tasks are difficult to perform perceptually, thus there are naturally significant gains to pre-computing the task results. The estimated time taken to process the task using the best design generated by our system (Design 1) compared to previous systems (Design 6) show significant timesavings with the latter time exceeding the former by approximately 4 times. The results from this task show that there are very clear gains that can be realized from considering data transforms in the automatic design process.

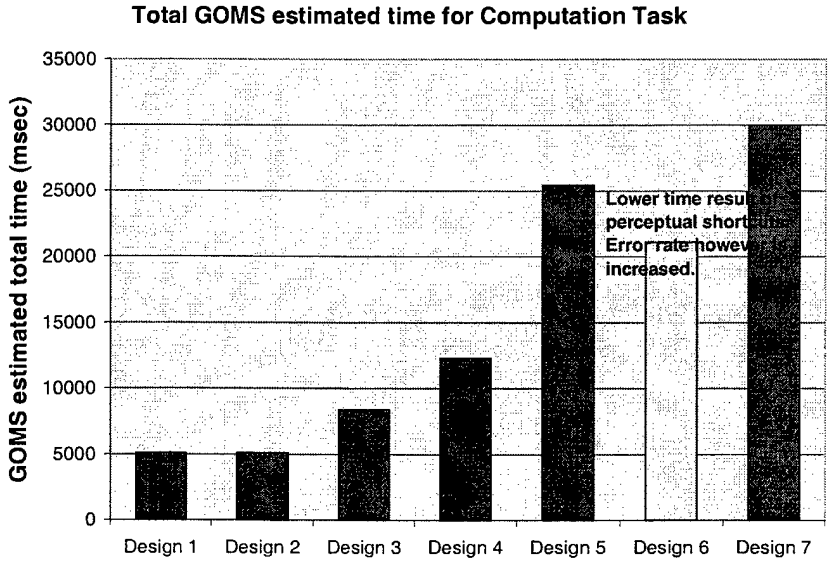


Figure E-4: GOMS estimated total time for Task 2 (computation task). The designs are ordered based on increasing cost on the *x-axis*. The *y-axis* shows the GOMS estimated total time in msec. All pink bars indicate pure mapping designs (i.e. designs that can be generated with current state of the art automatic design research). All other bars are designs made possible by work outlined in this thesis.

The advantages of data transform techniques can be further seen by comparing Design 1 which is the best design generated by our expanded designer and Design 6 which is the best design that can be generated by previous automatic designers. Design 1 is perceptually cleaner than Design 6, showing only the required computed results. Design 6 has six bars per university concept compared to the single bars per concept in Design 1. Our automatic design system chose Design 1 over Design 6 precisely because Design 1 utilizes fewer perceptual components (i.e. has less perceptual complexity), needs less display space, and requires no perceptual load to perform the task computation. On the other hand, in Design 6, the automatic system recognizes that significant perceptual processing is needed (to perform the subtraction and then the summation), in addition to the added complexity of showing 6 bars.

Other designs generated by our system for this compute task (not included in our GOMS analysis) internally process the task results to varying degrees of completeness. For example in Figure E-5, the total benefits are computed for the full and associate professor categories but not for the assistant professor category that must be computed perceptually. The summation task must also be performed perceptually.

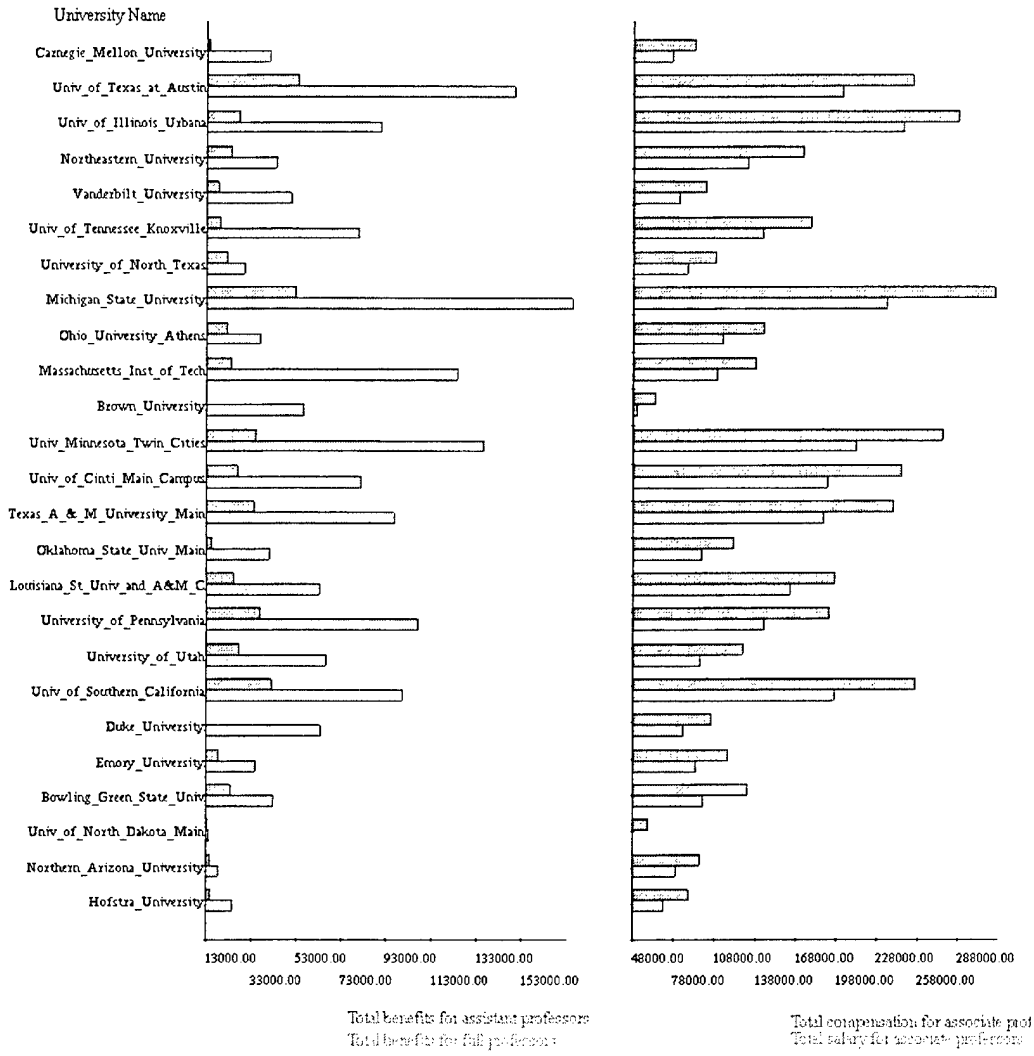


Figure E-5: Hybrid design where total benefits are pre-computed for full and associate professors (left chart) but not for assistance professors (right chart)

In Figure E-6, only the total benefits for full professors are pre-computed and all other operations must be performed perceptually. Note that to facilitate processing, there is a constraint in the designer that ensures all of the data attributes are mapped to the same graphical property class so that the subtraction and addition tasks can be performed more easily. In many of the examples here, the data attributes are all mapped to *x-position*. In chapter V we describe the constraint and cost structure in our automatic design system in greater detail.

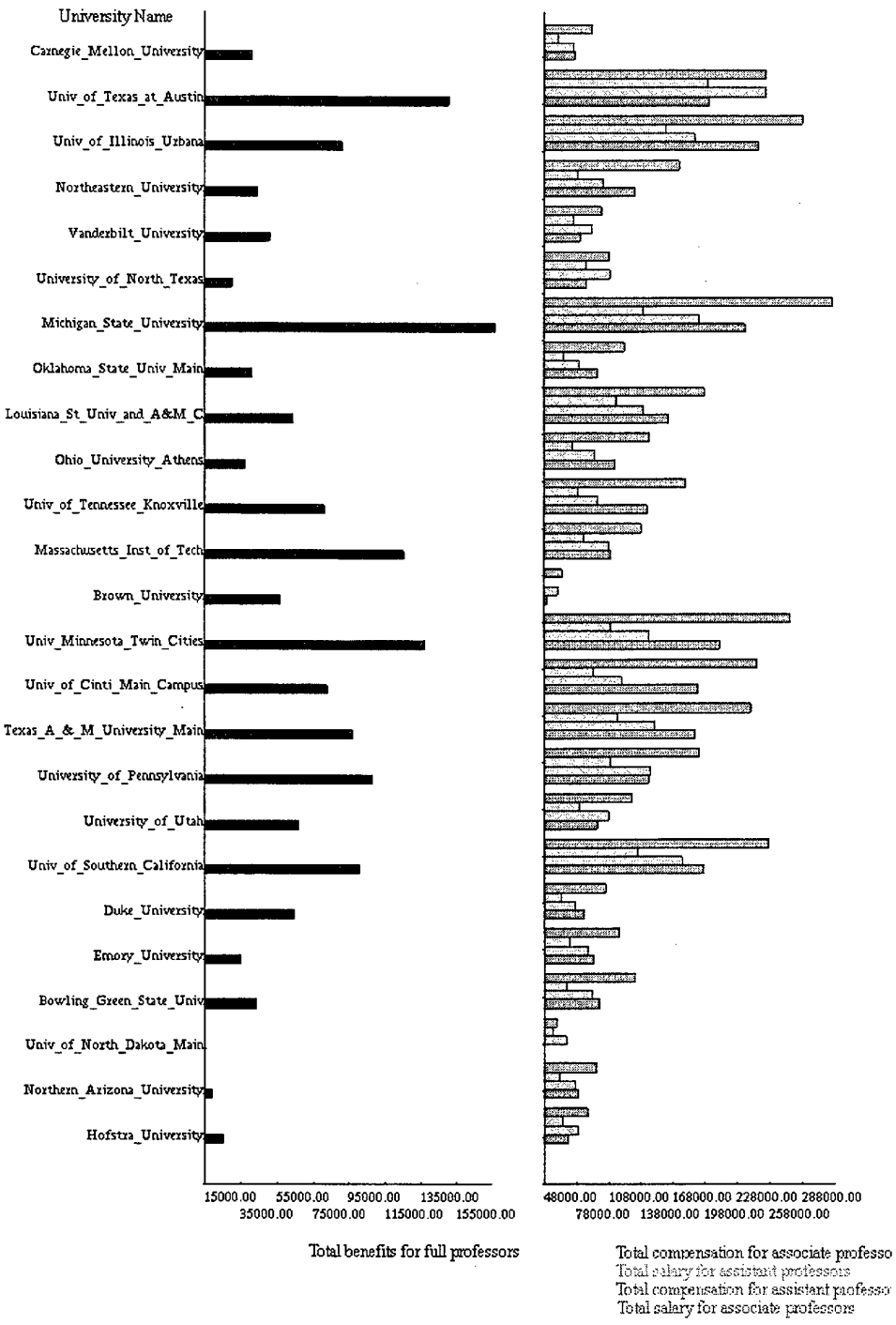


Figure E-6: Hybrid design where total benefits are pre-computed for only full professors (left chart) and the computation for associate and assistance professors (right chart) as well as the final summation task must be performed perceptually.

One may notice that if we had sorted or ordered the bars in the designs presented in this section based on *total_benefit* values, we would have been able to perform the given search task that we had set out for ourselves very easily. Specifically, we may augment our task specification with an additional *sort* task that encapsulates the *total_benefits* computation task as is shown below:

```
(compute   '(VALUE . SORT)
    (compute   '(VALUE . ADD)
            (compute '(VALUE . SUBTRACT)
                        (lookup  '(OBJECT . nil)
                          '(VALUE . AVG_FULL_COMPENSATION))
                     (lookup '(OBJECT . nil) '(VALUE . AVG_FULL_SALARY))
                     )
            (compute '(VALUE . SUBTRACT)
                     (lookup '(OBJECT . nil)
                             '(VALUE . AVG_ASSOC_COMPENSATION))
                     (lookup '(OBJECT . nil)
                             '(VALUE . AVG_ASSOC_SALARY))
                     )
            (compute '(VALUE . SUBTRACT)
                     (lookup '(OBJECT . nil)
                             '(VALUE . AVG_ASST_COMPENSATION))
                     (lookup '(OBJECT . nil) '(VALUE . AVG_ASST_SALARY))
                     )
            )
    )
```

Task E-3: Sort a set of universities by their total benefits

This addition causes some new designs to be generated, i.e. designs where the *sort* task is computed internally and only its results are shown (i.e. the *total_benefit* values are culled out). The designs that we analyzed and discussed above, however, will still all be candidates because they support the expanded task as well. It is important to note that this *sort* addition will only help us when we are searching for elements based on their ranking. If we were instead interested in comparing the relative total benefit values of two or more universities or in looking up the *total_benefit* values of particular universities, the *sort* task would be of no help. We expect that such additions or refinements to tasks will be a very common user operation. Users often do not have a clear enough idea of the tasks they want to perform that they are able to zero in on the correct and complete specification on first try. The same goes for communication. Users are rarely able to describe the exact information needed at the beginning of an information session. Realistically users will arrive at the information after having a series of conversations with the system where refinements and changes are made to the initial request. We discuss task refinement and its associated issues in section E-4.

E-3 Task 3: Comparison Task + Simple Computation
Evaluating the Relationship between State Size and Voting Results

In this final section we wanted to explore a comparison task. Upon further analysis, however, we found that a comparison task alone does not result in many more interesting designs or design issues over what has been previously discussed in the first two tasks. To make this example a bit more interesting and to start exploring task combinations, we included a simple computation task in addition to the comparison. In this example we compare the number of votes for three different political parties (*democratic, republican,* and *other*) in different states so that we may see how they rank with respect to each other. In addition to the ranking, we also want to see the total number of votes in each state so that we may determine the importance of a given victory. Larger states presumably carry a greater political gain.
The task specification entered into our automatic designer is shown below:

```
(compute '(VALUE . SORT)
         (lookup '(OBJECT . NIL) '(VALUE . DEMOCRATIC))
         (lookup '(OBJECT . NIL) '(VALUE . REPUBLICAN))
         (lookup '(OBJECT . NIL) '(VALUE . OTHER)))

(compute '(VALUE . ADD)
         (lookup '(OBJECT . NIL) '(VALUE . DEMOCRATIC))
         (lookup '(OBJECT . NIL) '(VALUE . REPUBLICAN))
         (lookup '(OBJECT . NIL) '(VALUE . OTHER)))
```

Compare the number of votes received by each of three political parties and determine their ranking. Also determine the total number of votes per state to ascertain the significance of any particular political victory.

From analyzing the displays generated by our system we were able to categorize the states into three groups. The larger states are generally carried by the *democratic* party, the medium to small sized states are generally carried by the *republican party* and finally the smallest states have an about equal mix between *republican* and *democratic* victories. The *other* party did not gain a victory in any of the states and in most cases ranked last. There is only one exception in "AK" where the *other* party ranked above the *democratic* party, and the *republican* party came in first.

As with the previous cases there are several different types of tasks that we may perform based on the general directions entered into our designer. We may for example be interested in the number of *democratic* victories compared to *republican* victories, we may be interested in finding abnormalities in ranking such as in the case of "AK" described above, we may be interested in examining the ranking of particular states that are of interest, etc. In the GOMS analysis below we wanted to examine a task that

utilized both the *ranking* and *total number of votes* information in tandem, so we chose the task of determining whether there is a relationship between *ranking* and *total number of votes.*

Ascertain whether there is a relationship between the party rankings and the total number of votes in a particular state.

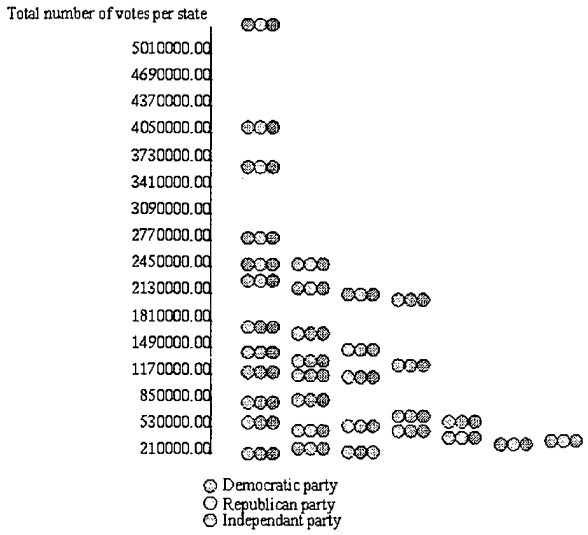
The following assumptions are made for all the GOMS procedures of this task:

Baseline Assumptions for Task 3 (Cmparison Task):

1. In this task there are three parties and in our task specification we stated our interest to see their respective rankings. Thus in all the graphics generated, all three parties are represented. In all the graphics generated, it is necessary to differentiate each of the parties so that we may identify a particular rank with a party name. Sometimes the encoding scheme used for the parties may be different from one design to the next. This is because our designer assumes each output design is separate and does not try to use similar encoding schemes across multiple alternative designs. In all of the GOMS procedures for this task we assume that the user is familiar with any encoding scheme used for the three parties. This assumption is reasonable since we are modeling an expert user. In addition, adding such information into the GOMS estimation does not change the rankings of the different designs because it adds a commensurate amount of time to each and every design.

E-3.1 Design 1

The first design generated is a purely computed design. Both *total number of votes* and *party rankings* are determined by the system and only the computed results are shown to the user. In the design below, each state is represented by a cluster of three marks. Each mark represents a different party, *red* represents the *democratic* party, *green* the *republican* party, and *purple* the *other* party. *Total number of votes* is mapped to the *y-axis* and the marks are ordered from left to right depending on the computed rankings of the different parties. The left-most mark is the party with the most votes and the right-most mark is the party with the least votes. The *x-axis* of this design does not encode any data. Clusters are however shifted to the right to improve layout and avoid occlusion problems. This graphic is especially efficient for performing the desired task because we can pre-attentively see the relationship between *ranking* and *total number of votes* within each *x-column* without having to attend to each *state* concept. Clusters high on the *y-axis* start off with a *red* dot (i.e. larger *states* are won by the *democratic* party), clusters in the mid to lower portion of the *y-axis* start off with a *green* dot (i.e. mid to small *states* are won by the *republican* party), and finally clusters at the bottom-most areas start off with either a *red* or *green* dot (i.e. the very small states are won by a mix of both *democratic* and *republican* parties).



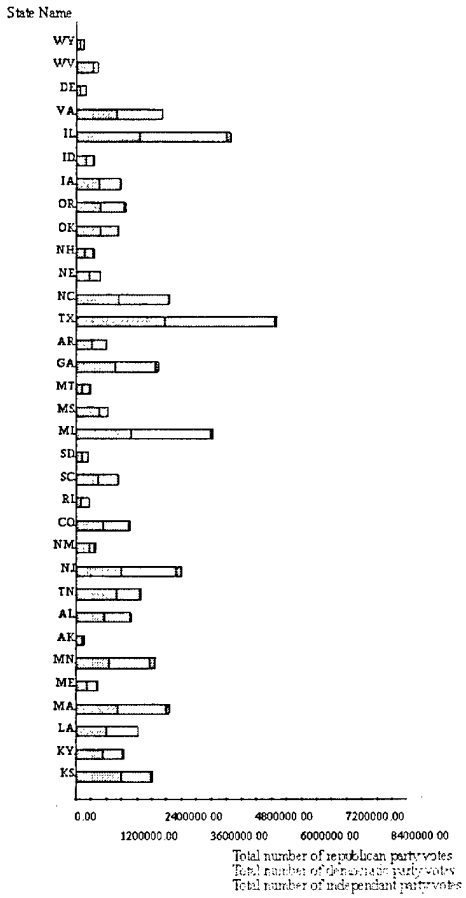
The GOMS procedure for this design is simple, requiring three perceptual groupings at the top, mid-to-low, and very low portions of the *y-axis*. Since the patterns can be viewed pre-attentively users can deduce the relevant information with just three perceptual scans.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Scan top of *y-axis* to see which color mark is to the right of the clusters there.	Attend scan top	50			*Red* or *democratic* party to the left.
	Initiate eye movement	50			
	Eye movement	30			
	Perceive information	100			
	Verify results	50			
	Sub-total		280		
This is pre-attentive for each *x-column* and there are 4 *x-columns* at the top so 4 * 280 msec are needed.	4 * 280			1120	
Scan to middle of *y-axis* to see which color mark is to the right of the clusters there.	This operation is similar to the one above, and there are 4 *x-columns* in the middle area so 4 * 280 msec are needed,			1120	*Green* or *republican* party to the left.
Scan to the lower y-axis to see which color mark is to the right of the clusters there.	This operation is similar to the one above, and there are 6 *x-columns* in the middle area so 6 * 280 msec are needed,			1680	*Purple* or *other* party to the left.
Total time				**3920**	

As can be seen from the GOMS sequence the design is simple and allows significant time savings because most of the data has been summarized due to the internal computation. In addition the visualization

design allows the relationship between *total number of votes* and *party ranking* to be viewed pre-attentively for each *x-column*, requiring only 14 perceptual processing steps. Note however that the visual design generated for this graphic is not a standard representation and as such the learning curve of using the design may be relatively high. However, since we assume expert users in our GOMS evaluation this issue has no effect on the total time. Also note that ranking of the three different parties is from left to right. This is based on the Western convention of reading which is from left to right. Users from other cultures may misinterpret this encoding scheme.

E-3.2 Design 2



We found it quite interesting and somewhat surprising that the second design generated in this task is a purely perceptual design. This is because the task entered is fairly complex one including a computation task which usually does not fare well with perceptual designs. However because the two tasks here (i.e. *summation* and *sort*) operate on the same set of data attributes (i.e. *number of votes for democratic, republican* and *other* party) a perceptual design that is effectively executed, can show the required data in an integrated fashion that supports both tasks as is the case below. In this design, the *number of votes* for each party type is mapped to the *x-length* of a different colored bar. The *green* bars show the number of

votes for the *democratic* party, the *red* bars show the number of votes for the *republican* party, and the *purple* bars show the number of votes for the *other* party. By mapping the *number of votes* information to the same graphical property class (i.e. *x-length*) we are able to facilitate both comparison and computation. By further stacking the bars, we greatly simplify performing the *summation* task perceptually. This design, however, is still only ordered second because more information must be perceptually mapped here including *state-name* and *number of votes* for each party (four attributes). This is in contrast to the two attributes (*total number of votes* and *party ranking*) that are mapped in the first design. In addition, despite the perceptual mapping choices used in the design that help facilitate the task, there is still some perceptual load that need to be expended to determine the task results especially for the comparison or ranking task which can be somewhat difficult because the bars are stacked.

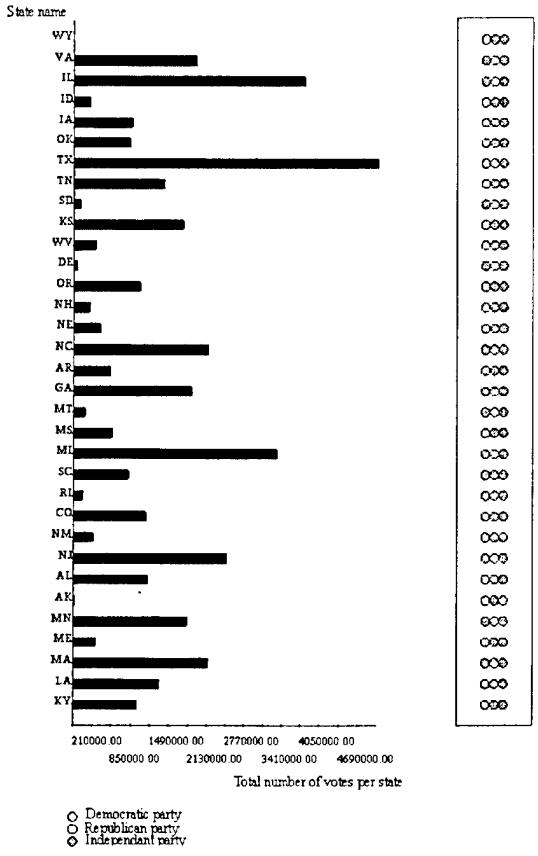
The GOMS procedure for this design begins with a scan for the longer bars in the design (i.e. the larger *states*) and for each long bar, we determine which color bar has the greatest length (i.e. which party is ranked first). Note that because of the data distribution, specifically the *number of votes* for the *democratic* and *republican* parties are usually very close in value resulting in almost equal bar lengths, it can sometimes be difficult to determine the ranking of the two parties. For better data distributions, it is possible to determine the rankings for groups of bars pre-attentively. However, in this case we at least need to attend to each bar and compare the *red* with the *green* lengths. In some states, where the values are especially close, we may not even be able to determine the ranking results accurately.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Scan for longest bars (pre-attentive) (then subsequently mid, and then short bars)	Attend scan	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive results	100			
			230		
For each bar cluster, determine which color segment is the longest. This may be a pre-attentive operation but because the *republican* and *democratic* votes are so close in value, it is difficult to make pre-attentive comparisons. As a result we assume that each bar must be attended to individually here.					
Compare the length of one colored bar with another	Attend compare	50			
	Initiate eye movement	50			
	Eye movement	30			
	Compare bar lengths	50			
	Verify results	50			
			230		
Note that since the *other* party usually has a very low number of votes we assume that the user is able to pre-attentively note this pattern initially, and thus only one pair of bar comparison is needed for each state. Thus the comparison cost for each state is 230 msec.					

In this procedure we need to scan for long, mid, and short bars, thus the estimated time for that is: 3 * 230 msec. In each of these operations we need to attend each bar in the set and determine the ranking of the parties. Since there are a total of 33 bars, the total time for determine the rankings of all bars in all three long, mid, and short bar categories is: 33 * 230 msec					
Total time	(3 * 230) + (33 * 230)			**7820**	

We want to point out that in the ideal case, where the data distribution allows pre-attentive (simultaneously) ranking of a set of bars, the total estimated time taken for this design would be significantly reduced at: (3 * 280) msec for perceiving each long, mid, and short bar category, and then another (3 * 280) msec for pre-attentively ranking each set of bars. The total time in this case is therefore only 1680 msec., which is much closer to the time taken in the first design.

E-3.3 Design 3



This design is a pure pre-computed design similar to Design 1. However, the data is separated over two different spaces and aligned based on *state-name*. The *total number of votes* within a state is mapped to the *x-lengths* of bars in the left chart and the *party rankings* are shown with a three mark cluster in the *table* to the right. As in Design 1 the marks are ordered from left to right (most votes on the left, least votes to the

right) and each party is represented by a different colored mark (*red = democratic, green = republican, purple = other*).

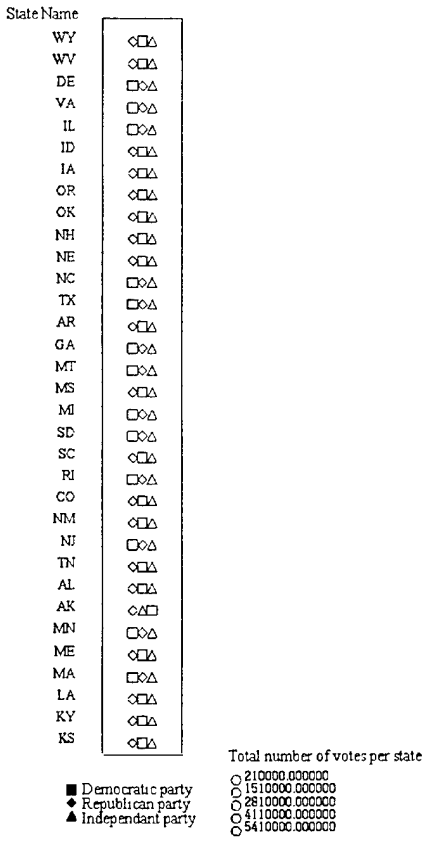
To solve the task using this design we first look for the states with a large number of votes on the left chart and then scan over to the right chart to lookup their rankings. We repeat these steps for the mid-sized and small states. Note that the ranking lookups are not pre-attentive and requires each of the relevant states in the large, mid, or small-sized categories to be visited.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Scan to next longest bar	Attend scan to next bar	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive bar	100			
	Sub-total		230		
Scan to the right and lookup ranking information and verify which party is ranked first.	Attend scan to right	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive marks	100			
	Verify results	50			
	Sub-total		280		
Total time to process a single row	230 + 280	510			
Total time	33 * 510			**16830**	

Note that the time taken to process this design can be reduced very significantly if we were to sort the elements on the *y-axis* based on *total number of votes* in each state. In doing this the design becomes very similar to Design 1 where the *y-axis* is used to encode *total number of votes* and as a result we are able to view the ranking results pre-attentively unlike the individual attention required in the GOMS sequence above. In this case, this design would take the same amount of time to use compared to Design 1. To automatically generate such a design however, requires the automatic system to recognize that the *y-axis* is used to encode and un-ordered (nominal) attribute, and since *position* allows ordering information to be shown, it is possible to add more information into the display without increasing perceptual complexity. Thus the designer must automatically transform a *nominal* data attribute into an *ordinal* attribute by adding ranking information into the element values. Such data characterization altering operations are beyond the scope of our work. Secondly, as was pointed out in the *summary* section of the previous task, this sorting operation requires an expansion to the current task specification that may cloud the main task(s) and associated design issues we are interested in exploring. We will however discuss the sort alternative in greater detail in section E-4.

E-3.4 Design 4

This design pre-computes the two task results as in Design 3, however, less effective graphical properties are used and it is for this reason that the design is ranked lower by our system. Specifically, *saturation* is used to show *total number of votes*, *shape* is used to show the three different parties, and *x-ordering* in each cluster is used to show ranking. Even though the previous design is less integrated, the results can be viewed with much greater accuracy and the additional time needed for the scan between the two spaces is not too significant. As with the previous case, ordering the *y-axis* based on *total number of votes* (i.e. *saturation*) can improve our ability to perform the task. We discuss this sorting issue in greater detail in section E-4.



The GOMS procedure for this graphic is very similar to that of the previous design. The difference is that instead of choosing elements based on *bar lengths*, here we are choosing elements based on their *saturation*. Initially we find all the saturated clusters, we then examine each cluster to determine how the parties ranked. We then repeat this process for the mid and low saturated clusters.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Scan to next most saturated cluster.	Attend scan	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive cluster	100			
	Sub-total		230		
Determine and verify shape of leftmost mark in cluster. (here we assume that the expert user does not need to refer to legend but are able to associate shape with party because of regular use of such graphics)	Attend get shape	50			
	Perceive shape	100			
	Verify results	50			
	Sub-total		200		
Total time	33 * (230 + 200)			**14190**	

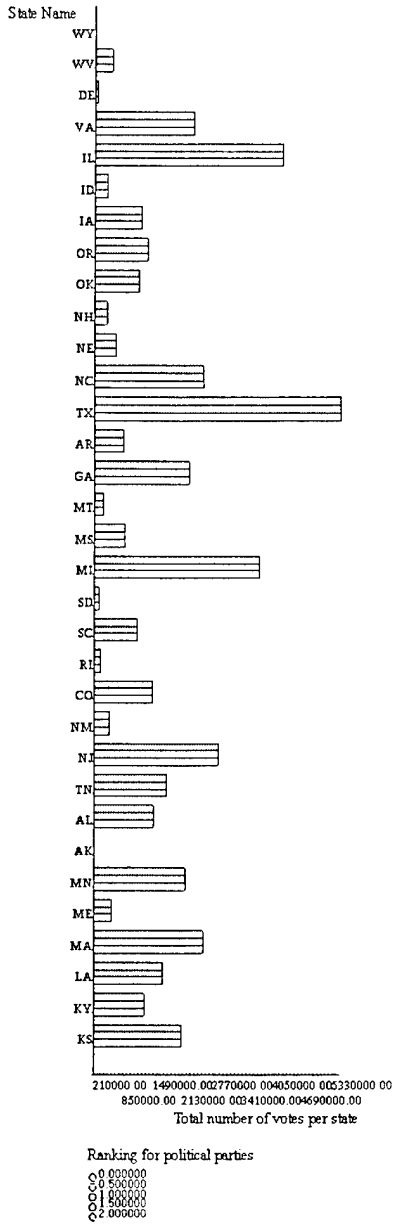
Note that the time taken for this task is slightly less than that of the previous design (2.5 seconds less). This is because the previous design requires additional eye-movements to scan from the left chart to the right table. However, the accuracy with which the task can be performed is not captured in the GOMS time estimation. However, since accuracy in grouping is less important to the task here, it has less of an effect than in the previous tasks where inaccuracies resulted in additional processing and inclusion of concepts that do not fulfill task requirements.

E-3.5 Design 5

This design is also fully pre-computed, it is interesting however, for the way in which it shows the computed results. *State* is shown on the *y-axis* and each *state* is associated with a cluster of three bars, one for each party starting with *democratic* on the top, followed by *republican* in the middle and *other* at the bottom. The *x-length* of all the bars encode the *total number of votes* within the state. The saturation of each bar represents the current ranking of the bar. Highly saturated bars are ranked higher compared to lower saturated bars. A primary weakness of this graphic is that we need to process each set of bars separately to get the most saturated bar (most highly ranked party) and then there is an additional step where we determine the position of that bar and associate its position to the relevant party. This is in contrast to the previous pre-computed designs where users can simply get to the highest ranked party by scanning to the left-most mark without having to even process any of the other marks in the cluster. Because of this the design is ranked lower by our system. Note that unlike the previous designs, even if we ordered the states on the *y-axis* based on *number of votes* we would still need to attend to each bar cluster individually.

In addition, unlike the *mark clusters* this graphic requires much more display area, due to the fact that a bar cluster expands the width needed to show each *state* concept three-fold. Also note that currently it is difficult to get the saturation values for bar clusters with small length (i.e. it is difficult to determine the party rankings for states with comparatively few total number of votes). For example consider "DE" and

"AK" where it is not possible to perceive the saturation values given the current *x-axis* scale. This problem however is not serious as it can be easily fixed by lowering the *minimum* range on the *x-axis*.



The GOMS procedure for this design begins with a search for all long bars (i.e. states with a large number of votes). For each of these bar clusters, we locate the most saturated object and determine whether it is on the top, middle, or lower position of the cluster. This will tell us which party was ranked first in the state. Subsequently we repeat this process for the mid-sized and short bar groups.

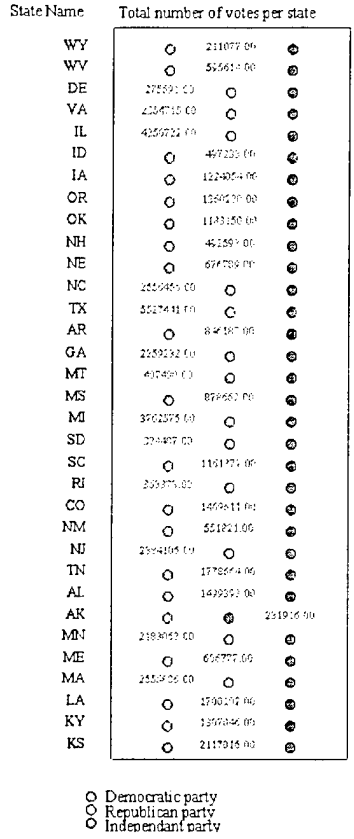
General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Scan to next longest bar	Attend scan to next longest	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive bar set	100			
	Sub-total		230		
Determine most saturated bar	Attend get most saturated	50			
	Compare all saturations and get most saturated bar (pre-attentive)	50			
	Verify results	50			
	Sub-total		150		
Get most saturated bar position within cluster	Attend get most saturated bar cluster position	50			
	Count Since the bar could be on the top, middle, or bottom positions the count could be anywhere from 1 to 3. On average we therefore assume a count of 2. Thus total count time is (2 * 50 msec) = 100 msec	100			
	Verify results	50			
	Sub-total		200		
Total time to process a single bar cluster	230 + 150 + 200	580			
Total time	33 * 580			**19140**	

As was expected, the time taken for this design is greater than other previous similar designs (i.e. Design 3 and Design 4). This is because additional steps are needed to compare the saturation values within each cluster and to subsequently identify the party associated with the most saturated bar. I.e. mapping *party ranking* to *saturation* is less efficient than mapping *party ranking* to *position*. The general structure of this design is also less traditional, thus learning time may also be greater.

E-3.6 Design 6

This design is also fully pre-computed, but it uses *labels* to represent the *total number of votes* in each state. In the design *state* is encoded on the *y-axis* and each state has a cluster of two marks and a label associated with it. The *red* label represents the *democratic* party, the *green* mark represents the *republican* party and the *purple* mark represents the *other party*. Ranking is shown based on the ordering of these three objects. The left-most object is the most highly ranked party, followed by the middle object, and finally the right-most object represents the party with the least ranking. While this label encoding provides accurate *total number of votes* figures, it is a very ineffective graphical property for grouping the states based on total votes because unlike all of the previous designs we need to attend to the label of each *state* concept.

Even worse yet, we need to first scan through all of the labels to get the range of values before we can begin the categorization process.



At the start of this GOMS procedure, we quickly scan through all of the labels to get the *min* and *max total number of votes*. During this process we assume that the working *min* and *max* numbers can be kept in STM (short term memory). Since we do not need to get exact *min* and *max* values here, it is reasonable to assume that users will round up or down the *total number of votes* figures to fewer significant digits to simplify comparisons as well as storage and recall from STM. Here we assume a general rounding to a single significant digit. Once we get the *min* and *max* numbers we go through the set of states again and only process those *total number of votes* entries that fall within the upper range of the *min-max* values extracted. When we are done, we repeat this process for the mid and lower range values. Thus we end up having to go through the table four times, once to determine the *min-max* values, and the next three times to process our three desired value groups namely high values, mid values and low values.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Scan down all numbers and get general *min* and *max* values to determine the range for the total benefit values.					
Time taken to process a single element. We assume that the *min* and *max* values can be stored in STM.	Attend check element to see if it is min or max candidate	50			*max* = *200k → 500k → 2mil → 4 mil → 5mil* *min = 200k*
	Initiate eye movement	50			
	Eye movement	30			
	Count digits in number in chunks of 3 digits to get general scale. Most of the numbers can be calculated in 2 chunks thus total time to count is: (2 * 50 msec) = 100 msec	100			
	Verify that digit chunks are consistent with scale of either current *min* or *max* (50 msec + 50 msec)	100			
	If chunks match, read left-most digit	290			
	Compare with *min* or *max* depending on digit chunks	50			
	Verify results and update *min* or *max* as necessary	50			
	Sub-total when chunks match		720		
Note that since all the numbers either fall within the *2 chunk (hundred thousand)* or *2+ chunk (million)* categories, we must always read the most significant digits to make the comparison with the current *min* and *max* values.					
Total time for processing all elements to get approx. min and max	33 * 720			23760	
Now we examine the time taken to process the ranking for each *total number of votes* category. Three categories: 1. States with > 1.5 mil. votes. 2. States with high hundred thousands of votes to 1.5 mil. 3. States with low hundred thousands of votes. Note that we do not charge any time for determining the bounds of these three categories because the time taken is difficult to estimate and we do not believe that it is significant because only approximate bounds are needed and no computation is necessary.					
Step 1: Process bars in each state size category beginning with the largest states followed by the mid and then small states.	Attend process states that are in current category	50			

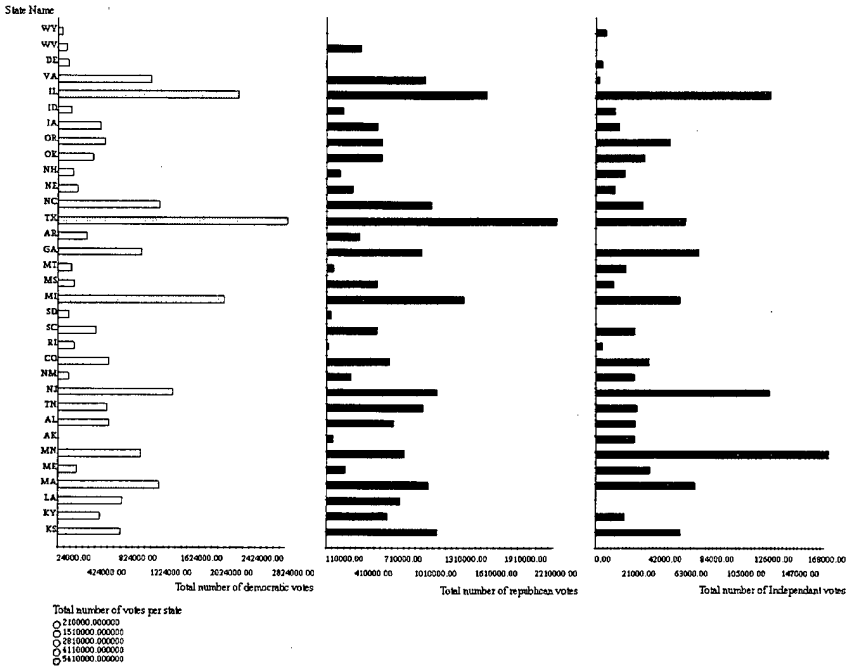
Step 2: Scan next number and determine whether it falls within the current category.	Attend determine next state that falls into current total benefits category	50			
	Initiate eye movement	50			
	Eye movement	30			
	Count chunks	100			
	Compare if number is within current desired category range based on chunk count	50			
	Sub-total		280		
	If so, read one or two most significant figures	290			
	Compare if number is within current desired category range	50			
	Verify results	50			
	Sub-total for entire step 2		670		
Step 3: If number is within current category bounds, then process ranking information. Specifically get the color of the left-most mark or label. The three objects are close enough in the display that an eye movement is not needed to get to the left-most mark.	Attend left-most mark color	50			
	Perceive color	100			
	Verify results	50			
	Sub-total		200		
Total time computation: There are three categories thus *Step 1* has to be performed 3 times = 3 * 50 msec. = 150 msec. *Step 2* has to be performed on all elements in the table three times (once for each *total number of votes* category). Some elements require reading of the most significant figure while others do not as they can be discarded with just the chunk count. In this particular data set, there are 13 numbers with 2 chunks and 20 numbers with 2+ chunks. Thus for category 1, we can discard 13 numbers based on just chunk comparison, and we have to fully process the other 20 numbers. Total time for category 1 = (13 * 280) + (20 * 670) = 17040 msec. For the second category we need to full process all numbers but we **do not** need to perform the chunk comparisons. Thus total time = 33 * 620 = 20460 msec. For the third category we discard 20 numbers and only fully process 13. Thus total time = (20 * 280) + (13 * 670) msec = 14310 msec. Total time to verify whether a state falls within its category for all three categories = 17040 + 20460 + 14310 msec = 51810 msec. *Step 3* is only performed once for each state because we assume that a state only ever belongs to at most one of the categories. Thus time taken = 33 * 200 msec. = 6600 msec.					
Total time	23760 + 150 + 51810 + 6600			**82320**	

Note that the time taken using this design is much higher than that of previous designs. This is because it is difficult to perform groupings of elements based on the *total number of votes* data attribute when it is mapped to *labels*. The same situation arose in Task 2, Design 5, where the estimated total time was higher than a subsequent lower ranked design (Task 2, Design 6). Higher costs are not assigned to *labels* because our automatic designer tries to balance the accuracy goal with the perceptual goal of being able to quickly identify patterns. While *labels* are not very effective for the latter goal, it does provide very accurate results. Since there is no preference in the current task specification one way or the other, our

designer balances both conflicting goals, with a stronger preference for the pattern identification goal. Refining the task with more complete accuracy preferences will help the designer adjust the weights between these two goals as necessary.

Another weakness of the design presented here with respect to the task is that it is difficult to determine where to set the category bounds based on *state* size. This is because the category bounds to some degree is based on the pattern in the *party ranking* data and we cannot process that information until after we have set up some temporary category bound values for grouping the elements. Thus we may have to readjust the category bounds as necessary mid-way in the analysis, and this will further lengthen the required time.

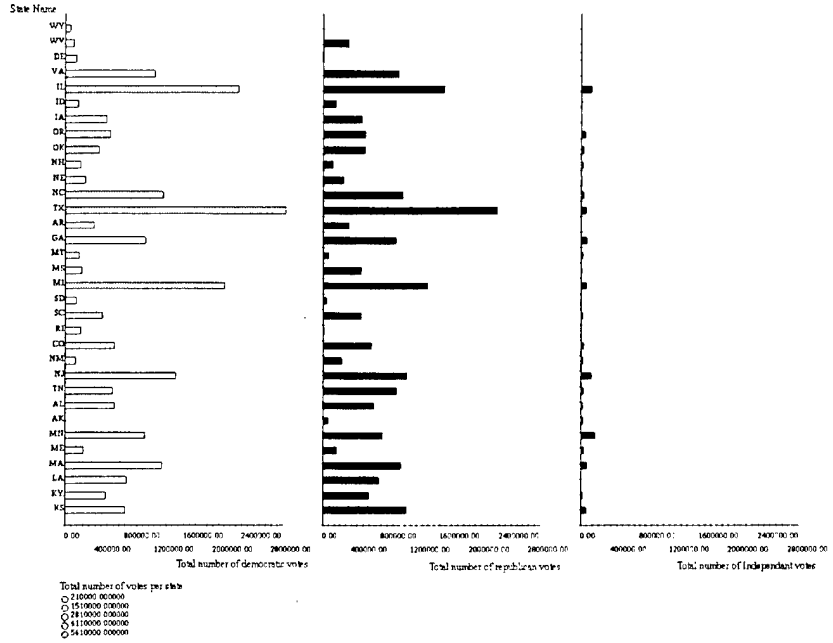
E-3.7 Design 7



This design is a hybrid design, with *total number of votes* pre-computed and mapped to *saturation*. The ranking is left to be performed perceptually by the user. Specifically *number of democratic votes* is mapped on the *x-lengths* of the left-most chart, *number of republican votes* is mapped on the *x-lengths* of the middle chart, and *number of other votes* is mapped on the *x-lengths* of the right-most chart. This design is ranked low for several reasons. The primary reason is because the information is spread out over many different spaces and this lack of integration as we have shown previously increases perceptual load and forces users to perform comparisons of values across different charts which can be difficult. Secondly, more data attributes are shown here compared to most of the previous designs. In this design five data attributes are mapped including *total number of votes, number of democratic votes, number of republican*

votes, number of other votes, and *state-name*. Finally, the computed *total number of votes* is mapped to *saturation*, and *saturation* is not a very accurate graphical property for showing continuous values.

As was the case in Task 2, Design 4 and Design 7, having consistent axis scales here for the three charts can facilitate the *comparison* task. Thus we start our GOMS procedure with getting the combined *min* and *max* values for the three charts (0.0, and 300000) and then altering the required axis ranges so that all three charts share identical *min* and *max* values. This altered chart design is shown below.



Before we begin the task, we note that all of the bars in the rightmost chart are very short compared to the two other charts and as a result they can be discounted from our subsequent comparison operations. To perform this task, we start processing the left-most chart and search for bars that are highly saturated. For each highly saturated bar, we scan to its rightmost edge and note its length. We then scan to the right and compare its length with the corresponding bar on the middle chart. Note that for bars that have similar lengths, it may be difficult to accurately compared their lengths especially across different graphical spaces. For such cases, we can estimate the length of a bar using two fingers, and then move our hand over to the related chart and compare our finger interval with the corresponding bar in the chart. This allows us to more accurately maintain perceptual state (i.e. *bar length*) across charts and as a result helps us perform more accurate length comparisons. Another alternative would be to lookup the actual number of votes of the bars from their respective axes in order to get an accurate comparison result. This technique is even more accurate than the finger interval processing method, but it is cognitively taxing and requires much more processing time. Below we estimate the time taken for both these cases.

General goal	Cognitive, perceptual, or articulatory step taken by user	Time taken (msec)	Sub total (msec)	Total time (msec)	Target objects (data or graphics)
Total time taken to get consistent axis min-max values for the three charts.	Time taken from Task 2, Design 7	3670			0.0, 300000
Click on first min or max value to change	Time taken from Task 2, Design 7	370			
Type in min/max value	Attend input in the min value	50			
	Initiate move hand to numeric keypad	50			
	Move from mouse to numeric keypad	132			
	Attend type value	50			
	Type in value: Number with move required takes 40 + 60 + 60 = 160 msec. (move, upstroke, downstroke) Number with no move required takes 60 + 60 = 120 msec. (upstroke, downstroke) Value 0 takes (1 * 160) = 160 msec. Value 300 000 takes (2 * 160) + (4 * 120) = 800 msec.	160/ 800			
	Verify results	50			
	Sub-total for min entry		492		
	Sub-total for max entry		1132		
Move hand back to mouse	Time taken from Task 2, Design 7	182			
Total time to actually enter in the proper min-max values includes: 3. Time taken to enter in the min values = 2 * (370 + 492 + 182) = 2088 msec. 4. Time taken to enter in the max values = 2 * (370 + 1132 + 182) = 3368 msec. Total time for entry = 5456 msec.					
Now that we have re-scaled the three charts, we proceed to processing the bars.					
Process bars based on decreasing saturation	Attend get next most saturated bar	50			
	Initiate eye movement	50			
	Eye movement	30			
	Attend get bar length	50			
	Perceive bar length	100			
	Sub-total		280		
Scan to same row over to the right and get length of bar there	Attend get length of bar to the right	50			
	Initiate eye movement	50			
	Eye movement	30			
	Perceive bar length	100			
	Sub-total		230		

Compare the two bar lengths	Attend compare	50			
	Compare	50			
	Verify results	50			
	Sub-total		150		
Time taken to process one bar row	280 + 230 + 150	660			

As was noted previously, in some cases the bar lengths may be too close in value, and it is difficult to compare their lengths accurately across different graphic spaces. In this case, we have two alternatives:

Case 1: We use our fingers to mark off the length of the first bar on the left chart and then move our hand over to the bar on the right, comparing the length between our fingers with the bar length.

Case 2: We lookup the number of votes corresponding to the given lengths from the *x-axis* and then compare these values.

CASE 1:

Time taken to process a single row of bars for Case 1:

In this case we assume that the user increases accuracy of comparison across charts by using their fingers to approximately capture the bar lengths and to transfer this state across charts. To facilitate finger pointing etc, we assume that the chart height takes up approximately an entire 21 inch display screen, with a height of approximately 11 inches. Since there are 33 bars, each bar height is approximately 0.33 inches. We also observe that the data distribution results in an average move of about 6 steps from one bar to another which approximates to 6 * 0.33 = 2 inches.

We will also assume that the width of the visualization takes up the entire width of the display screen, measuring at approximately 14 inches. Each chart is therefore approximately 4.5 inches. Average length of a bar is approximately a third of the chart width thus average bar length = 4.5 /3 = 1.5 inches.

Move hand to appropriate bar. Total time to move hand based on Fitts Law = 100 * ($\log_2$ (2.0/0.33)+.5) = 271 msec.	Attend move hand to current bar being processed	50			
	Initiate hand move	50			
	Hand move	271			
	Sub-total		371		
Move fingers simultaneously to the beginning and end of bar. Assume that the range of area at bar end-points = 0.05 inches Total time to move fingers using Fitts Law = 100 * ($\log_2$ (0.75/0.05)+.5) = 395 msec. Note that perceiving the end points of the bar is taken into account in Fitts Law.	Attend fingers to begin and end of bar	50			
	Initiate move	50			
	Move fingers	395			
	Sub-total		495		
Move hand to left chart exactly over the relevant bar. Total time to move hand using Fitts Law = 100 * ($\log_2$ (4.5/0.33)+.5) = 382 msec.	Attend hand move	50			
	Initiate hand move	50			
	Hand move	382			
	Sub-total		482		
Compare width indicated by finger with width of current bar	Attend compare	50			
	Compare	50			
	Verify results	50			
	Sub-total		150		
Total time taken to process a single row for CASE 1	371 + 495 + 482 + 150	1498			

CASE 1: Total time includes: Time taken to initially process all bar pairs = 33 * 660 msec. = 21780 Time taken to process all the bar pairs with similar lengths. In the design above we found 17 such bars thus the total time for this operation = 17 * 1498 msec. = 25466 msec.					
Total time taken for entire task assuming CASE 1 = Time taken to determine min-max values for 3 charts (3670) + Time taken to re-scale all chart axes (5456) + Time taken to process bars (21780 + 25466)					
Total time taken for task assuming CASE 1	3670 + 5456 + (21780 + 25466)			**56372**	
CASE 2: Time taken to process a single row of bars for Case 2:					
Lookup number of democratic votes for current bar row	This time value was taken from the computed time to lookup values on axis in Design 4, Task 1	1135			
Lookup number of republican votes for current bar row.	This time value was taken from the computed time to lookup values on axis in Design 4, Task 1	1135			
Compare the two vote numbers to see which is higher. Here we assume that in the previous lookups the numbers are rounded up to two significant figures. This is because it is difficult to keep longer numbers in STM and it is difficult to compare such numbers as well. In this example it is sufficient in all cases to process the numbers up to two significant figures.	Attend compare	50			
	Compare first digit	50			
	Compare second digit	50			
	Verify results	50			
			200		
Total time taken to process a single row	1135 + 1135 + 200	2470			
CASE 2: Total time includes: Time taken to initially process all bar pairs = 33 * 660 msec. = 21780 Time taken to process all the bar pairs with similar lengths. In the design above we found 17 such bars thus the total time for this operation = 17 * 2470 msec. = 41990 msec.					
Total time taken for entire task assuming CASE 2 = Time taken to determine min-max values for 3 charts (3670) + Time taken to re-scale all chart axes (5456) + Time taken to process bars (21780 + 41990)					
Total time taken for task assuming CASE 2	3670 + 5456 + 21780 + 41990			**72896**	

Not surprisingly the time taken for case 2, which only requires perceptual and motoric loads is lower than the time taken for case 1, which requires cognitive computation and comparison. It is also interesting to note that the time estimated here for both cases is less than the estimated time for the previous design. This is made possible by the nature of the data distribution and the specific nature of the task we chose for our GOMS evaluation. Specifically, significant time savings were realized here due to the fact that we did not need to process information from the third chart at all.

Note that in the general case where all the information must be processed, the total time taken for both cases becomes significantly higher and exceeds that of Design 6. We show the modified computations for complete processing of all chart data below. The time taken to initially process the bars are increased by 230 msec + 150 msec per row from having to scan to and make comparisons with bars in the third chart. In addition, the time taken to process bar pairs with similar lengths is also increased based on additional processing with the right-most chart. We assume that the data distribution is similar to that of the previous charts, thereby requiring double the number of similar length comparisons.

CASE 1: Total time includes: Time taken to initially process all bar pairs = 33 * (660 + 230 + 150) msec. = 34320 msec. Time taken to process all the bar pairs with similar lengths. In the design above we found 17 such bars thus the total time for this operation = 17 * (1498 + 495 + 482 + 150) msec. = 44625 msec.					
Total time taken for entire task assuming CASE 1 = Time taken to determine min-max values for 3 charts (3670) + Time taken to re-scale all chart axes (5456) + Time taken to process bars (34320 + 44625)					
Total time taken for task assuming CASE 1 and full processing of ALL data	3670 + 5456 + (34320 + 44625)			**88071**	

CASE 2: Total time includes: Time taken to initially process all bar pairs = 33 * (660 + 230 + 150) msec. = 34320 msec. Time taken to process all the bar pairs with similar lengths. In the design above we found 17 such bars thus the total time for this operation = 17 * (2470 + 1135 + 200) msec. = 64685 msec.					
Total time taken for entire task assuming CASE 2 = Time taken to determine min-max values for 3 charts (3670) + Time taken to re-scale all chart axes (5456) + Time taken to process bars (21780 + 43690)					
Total time taken for task assuming CASE 2 and full processing of ALL data	3670 + 5456 + (34320 + 64685)			**108131**	

E-3.8 Summary

As with the previous two tasks, Figure E-7 shows that in most part the ranking assigned by our automatic design system is consistent with the GOMS estimated total times. In Figure E-7 there are two bars shown for Design 7, and one of these bars (purple bar) has a lower estimated total time compared to Design 6. However as was discussed previously this was only made possible because in Design 7 we were able to solve the task without considering any of the information in the right-most chart. In this example the data distribution is such that the information in the right-most chart has no effect on the results of the party-ranking task. When we estimate the time taken to process all of the information, the total time for Design 7 exceeds that of Design 6 as is shown by the second (right, green) bar. It would be very challenging and

interesting to characterize this class of perceptual shortcuts and encode that information as heuristics into our automatic design system. However, this issue is complex and we leave it for future work.

Like the previous computation task, most of the top designs for this comparison + simple compute task are pure data technique designs or hybrid data + mapping designs. Out of the 19 visualization alternatives generated, 5 were pure data technique designs, 12 were hybrid data + mapping designs, and only 2 were pure mapping designs. Thus most of the top designs for this task (89.5 %) are only made possible because of the design space expansion from adding data techniques into the automatic design process.

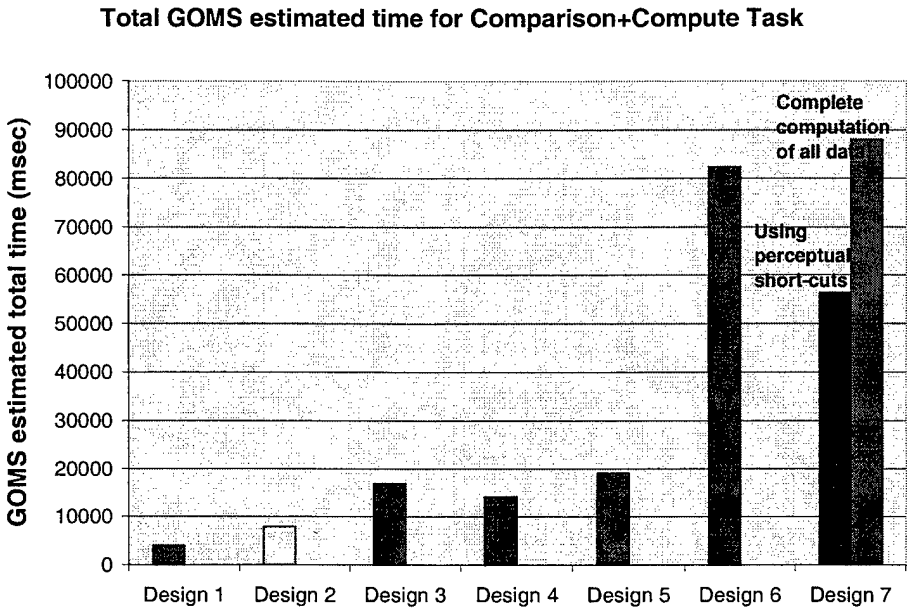


Figure E-7: GOMS estimated total time for Task 3 (comparison task). The designs are ordered based on increasing cost on the *x-axis*. The *y-axis* shows the GOMS estimated total time in msec. All pink bars indicate pure mapping designs (i.e. designs that can be generated with current state of the art automatic design research). All other bars are designs made possible by work outlined in this thesis.

An interesting feature in Figure E-7 is that the design ranked second is a pure mapping visualization. This is in contrast with the previous computation task where the best pure mapping design was ranked 6th. There are several reasons why the pure mapping design (Design 2) performs well here. First of all, the two sub-tasks in the specification, i.e. the *summation* and *sort* tasks, both operate on exactly the same data attributes (i.e. number of votes for each of the three parties). In addition, both tasks are facilitated by similar mapping constraints (e.g. mapping all the data attributes to the same graphical properties, in this case, *x-position*). Because of their shared data attributes and task constraints we are able to use the same graphical objects to solve both tasks as in Design 2. In addition, the computation task in this example is just a simple *summation* that can be effectively represented by using stacked bars. Thus the nature of the two

tasks allows for an effective perceptual mapping to be achieved. However, these perceptual requirements are so tight that there are few designs that are able to fulfill all of them. Hence other than Design 2, there are no other pure mapping designs in the top spots.

Despite the ability of Design 2 to combine the perceptual goals of both the tasks so well, the top design in this example is still a data transform design with both *total number of votes* and *party-ranking* pre-computed. While both Design 1and Design 2 have the same number of objects, the former requires fewer data attributes to be mapped (i.e. only 3, namely *total number of votes, party ranking* and *party type*) compared to the five attributes in Design 2 (*number of votes for democratic party, number of votes for republican party, number of votes for other party, state-name* and *party-type*). In addition the comparison task may sometimes be difficult to perform on Design 2 because of the bar stacking and the narrow differences in values between the number of *democratic and republican* votes. In contrast the ranking results in Design 1 require very little perceptual effort and are accurate. For these reasons our automatic designer chose Design 1 over Design 2.

As with the previous task, some of the designs here can be made more effective by refining the task so that we sort the *states* based on their *total number of votes*. This sorting makes it unnecessary or very simple to perform search and categorization tasks. We discuss task refinements in the next section and how it relates to the automatic visualization design process.

E-4 Task Refinement and Sorting

As was discussed previously it is rarely the case that a task is completely and accurately specified at the start of a data analysis or communication session. Users often refine their task or communication goals through an iterative process where they consider output from the computer system, integrate that output with their current task model, change that model as necessary, and finally convey those changes to the computer system. At the end of this cycle, the iterative process begins anew. For example, consider Task 2 (computation of total benefits) and Task 3 (comparison of party votes and computation of total votes). In both these cases, once we analyzed the initial output designs, we recognize that our ability to solve the tasks can be enhanced by refining our original specification to include a *sort* operation. This is because in both these cases we were interested in finding or grouping objects based on a single attribute (*total benefits* attribute in Task 2 and *total number of votes* attribute in Task 3). By sorting the objects in the display based on the relevant attribute we can quickly perform the task with less perceptual load.

In the original tasks (before the *sort* addition) it was not made clear to the design system what the user intended to do with the *total benefit* values in Task 2 or the *total number of votes* values in Task 3. The user's intention could have been to compare the *total benefit* ratios among several universities, find universities with particular *total benefit* values, lookup the *total benefits* for given universities, group universities based on *total benefits,* etc. According to Bertin [*Bertin, 1981*], tasks can be divided into three

levels of reading: elementary, intermediate, and global. For elementary tasks it is more important to be able lookup single values clearly and accurately, while for the intermediate and global tasks it is more important to be able to get a sense for general gestalt patterns or trends in the data set. When no information is given with respect to the level of reading desired (e.g. what to do with the *total benefit* values in Task 2 once computed), our designer tries to make design decisions that balance between the three different levels, as was done in previous systems. Essentially the higher levels of reading (e.g. finding data patterns) are given preference over elementary readings, unless the user explicitly specifies a desire for the latter.

In these Task 2 and Task 3, the big win with expanding the specification with a sort operation is not necessarily the pre-computation of the sort but rather the ability to integrate the sort results into the graphic effectively while reducing perceptual complexity. This can be achieved by recognizing that when a nominal attribute (e.g. *university name* or *state name*) is mapped to *position* we are not utilizing all the expressive power of the *positional* graphical property. Specifically *positionals* can capture order or ranking information as well. Thus it is possible to show the ranking results by integrating it into an originally *nominal, unique* attribute (e.g. *university name* or *state name*) and turning that *nominal* attribute into an *ordinal*. In this way, additional information is included into the graphic without increasing the number of graphical properties or objects in the visualization.

University Name

Michigan_State_University
Univ_of_Texas_at_Austin
Univ_Minnesota_Twin_Cities
Univ_of_Southern_California
University_of_Pennsylvania
Texas_A_&_M_University_Main
Massachusetts_Inst_of_Tech
Univ_of_Cinti_Main_Campus
Univ_of_Illinois_Urbana
Univ_of_Tennessee_Knoxville
University_of_Utah
Louisiana_St_Univ_and_A&M_C
Northeastern_University
Bowling_Green_State_Univ
Duke_University
Ohio_University_Athens
Vanderbilt_University
Brown_University
Oklahoma_State_Univ_Main
Emory_University
University_of_North_Texas
Carnegie_Mellon_University
Hofstra_University
Northern_Arizona_University
Univ_of_North_Dakota_Main

Figure E-8: List of universities ranked based on their computed total-benefit values

For example Figure E-8 shows a visualization design which lists the university names based on their *total-benefit* values. This design fully pre-computes the *sort* augmented, total benefits *computation* task and adds the ranking information to the university names, which are mapped to *y-position*.

This design is more effective than the best design in Task 2 because no perceptual load is required to identify the four longest bars as was the case in Design 1, Task 2. Instead users can just read off the first four names on the sorted list. Note that mapping the sort results in any way other than the *nominal-to-ordinal* method described above does not result in any clear processing benefits. This is because the sort results are a quantitative attribute very similar to the total benefits attribute. By computing and showing the *sort* results instead of the *total benefit* results we are essentially replacing a quantitative attribute with another. For example in Figure E-9 ranking information is mapped to *mark x-position* while in Figure E-10 *total benefit* values are mapped to bar *x-lengths* as in Design 1, Task 2. Both the designs take approximately the same amount of time to process and their GOMS procedures are almost identical. The only time when Figure E-9 is possibly superior is when there is data dwarfing.



Figure E-9: Visualization design where the total benefit values are sorted and the sorted rankings are mapped to *x-position*

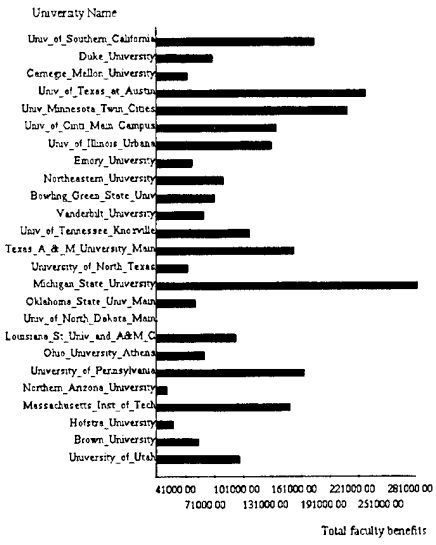


Figure E-10: Visualization where the total benefit values are mapped to bar *x-lengths*

Another interesting aspect of this example is that sorting can help highlight certain types of data trends and patterns. I.e. sorting can support certain tasks with intermediate and global reading level requirements. In such cases it becomes less important to encode the sorted attribute values with graphical properties that facilitate these higher levels of reading. Instead we can give a greater preference to the elementary reading levels and encode the attribute to facilitate lookup accuracy. For example, rather than

ordering the universities top down based on benefit values and at the same time showing the *total benefit* attribute using *position* (as in Figure E-11), it may be more beneficial to map *total benefits* to *text* instead (as in Figure E-12). This is because sorting supports some of the higher level reading tasks and text labels are a good complement for providing support to the lower reading levels.

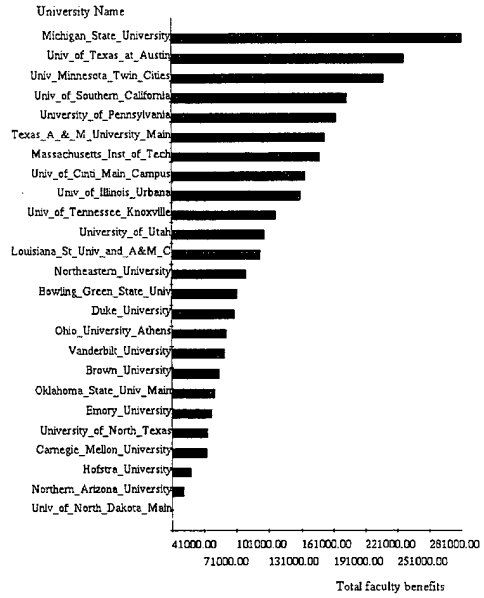


Figure E-11: Visualization where the universities are ordered on the *y-axis* based on *total-benefits* and in addition the *total-benefit* values are mapped to *bar-lengths*.

University Name	Total faculty benefits
Michigan_State_University	320788.00
Univ_of_Texas_at_Austin	265209.00
Univ_Minnesota_Twin_Cities	245474.00
Univ_of_Southern_California	209920.00
University_of_Pennsylvania	199225.00
Texas_A_&_M_University_Main	188679.00
Massachusetts_Inst_of_Tech	183356.00
Univ_of_Cinti_Main_Campus	169492.00
Univ_of_Illinois_Urbana	164436.00
Univ_of_Tennessee_Knoxville	140956.00
University_of_Utah	129774.00
Louisiana_St_Univ_and_A&M_C	126133.00
Northeastern_University	112416.00
Bowling_Green_State_Univ	103558.00
Duke_University	101040.00
Ohio_University_Athens	92852.00
Vanderbilt_University	91711.00
Brown_University	86175.00
Oklahoma_State_Univ_Main	82513.00
Emory_University	79007.00
University_of_North_Texas	75046.00
Carnegie_Mellon_University	74240.00
Hofstra_University	59244.00
Northern_Arizona_University	52341.00
Univ_of_North_Dakota_Main	40730.00

Figure E-12: Visualization where the universities are ordered on the *y-axis* based on *total-benefits* and in addition the *total-benefit* values are mapped to *text labels*.

We want to point out that the sorted designs (e.g. Figure E-8, Figure E-11, and Figure E-12) cannot be generated with current state of the art automatic systems because no pre-processing (i.e. data transformation) operators are considered in their design process. Thus, this example further underscores the importance of integrating data operations into automatic design because as we saw in Figure E-8, Figure E-11, and Figure E-12 pre-processed sorts can improve design effectiveness in interesting ways. In our system we are able to pre-compute the sort task but we currently **do not** allow operations that can alter the basic characterization of data attributes. I.e. we do not allow the ranking information to be added into *university name* by converting it into an *ordinal*. We suspect that there is a whole class of data alteration opportunities and thus we leave fuller treatment of this issue for future work.

In the *sort* example above, we present a simple task refinement operation, where a new task is added, encapsulating one of the previous tasks. Recall that in appendix C-5, we presented a *car purchasing* example where more extensive and complex task refinements were made. Currently our designer does not

explicitly support the task refinement process. Each design requested is assumed to be unrelated and thus the designer does not strive to maintain any visual consistency between one request to the next. This visual consistency issue is an important one and it is currently being explored by other researchers in the field. We hope to integrate such work into our designer in the future. Of course the automatic design process implicitly supports the task refinement process by presenting users with a set of alternative visual designs for solving the same tasks. By analyzing and comparing these designs, the important elements of the user's task may become clearer and this would in turn help in subsequent task refinements.

E-5 Why GOMS?

The primary reason why we chose GOMS as an evaluation method for our work is because it would take a significantly longer amount of time to perform the same tests with users. In addition by mapping out the minute cognitive, perceptual and motoric steps required in a GOMS analysis we are able to better analyze the particular strengths and weaknesses of visualization designs and identify where most of the processing time is being spent. Some of the weaknesses of the GOMS analysis method used here, however, is that we only model the performance of experts and it is difficult in many cases to account for task accuracy problems, readability problems (such as occlusion, graphic density, and dwarfing) as well as more complex usability issues such as user experience, user fatigue, number of errors performed by users, task solution formulation (time taken to come up with a solution on how to solve the task with the graphic at hand), etc. While it would be interesting and useful to collect this type of information, they are **not required** for testing the three major issues we want to evaluate in our system. Specifically we want to determine the following:

- Our theories can be implemented and they perform as expected. I.e. the design metrics and heuristics used in the designer result in output designs that are ordered according to complexity of use (by "complexity of use" we refer to cognitive, perceptual, and motoric complexity).
- Our work increases the breadth of designs that can be generated by automatic systems. I.e. our automatic system should be able to produce designs that cannot be previously generated.
- Our work improves the effectiveness of visualizations generated. I.e. the expanded design space contains visualizations that allow certain task classes to be solved more effectively.

As was previously specified these goals can be adequately evaluated using GOMS because first of all we are only interested in testing the quality of our design with designs that can be generated by current state of the art automatic systems. We do not make any claims with respect to the quality of designs generated by our system and those that are generated by a human designer. Secondly all of the GOMS evaluations are generated based on the same set of assumptions and estimate time measurements. All procedures are also modeled based on expert performance and this consistency in the evaluation method makes the time comparisons among the different GOMS analyzed designs more equitable because any change in assumptions will affect all of the GOMS procedures in consistent ways. Finally and most importantly we

do not require accurate time estimates for this evaluation. To ensure the correctness of our system it is only necessary for us to identify groups of designs that have similar time estimates and ensure that these groups are ordered properly by our design system. I.e. the absolute time estimates are less important, what is more important are the large time differences between the different design groups and our analysis of what are causing these differences.

We also stress that while it is possible to evaluate designs in our automatic system by generating GOMS sequences, this is a very expensive process and can cause many complex issues to arise when applied to partial designs. Single small changes to partial designs may cause a significant change in the GOMS sequence used and may invalidate a previously forecasted sequence. This would in turn result in great swings in the time estimations. In addition as we have discussed in this section GOMS does not take into account certain important issues such as accuracy and readability problems. Thus we have opted to use a higher level more abstract cost system in our designer based on the heuristics and metrics presented in chapter IV. Details of this cost structure are presented in chapter V. In this appendix however, we proved the correctness and feasibility of our higher level cost structure by showing that our designer orders its output designs consistently with GOMS time estimations.

E-6 Conclusion

In summary the GOMS analyses in this section validates the three evaluation goals we set out to test. Specifically, we showed that:

- The output order of our automatic designer (i.e. our design heuristics used) does indeed conform to cognitive, perceptual, and motoric complexity as computed by GOMS. I.e. the theoretical concepts developed here for characterizing and expanding the visualization techniques design space for automatic visualization generation can be implemented and the results are meaningful (i.e. conforms to GOMS computed times).
- Adding data transform techniques into the automatic design process expands the visualization design space and enables whole new sets of interactive and non-interactive visualizations to be generated. Many of these designs are shown in the following sections.
- Some of the new designs generated as a result of work developed in this thesis (i.e. pure data transform techniques and hybrid data + mapping designs) perform much better than the designs that can be generated with current state of the art technology (i.e. pure mapping designs) for the task classes we considered (*computation, search, comparison*); with the highest gain in computation tasks.

Demonstrating these three goals with GOMS validates the thesis statement set out in this document. This GOMS evaluation also attests to the generality and usefulness of the rules employed in our automatic design system and places our cost structure on a concrete, proven, empirical basis of cognitive, perceptual,

and motoric steps. The analyses and design examples presented here also underscore the importance and richness introduced by data transform techniques to the visualization design process.

Throughout this appendix we have also highlighted important issues and analyzed why our design system ranks certain visualizations over others. More details are given on our designer ranking algorithms in the chapter V where we describe the implementation details of our system. Some of the important issues brought forth in this evaluation appendix help scope out the areas that we deal with in our work and the areas that we don't. Some of the interesting challenges that we leave for future work include the problem of ensuring visual consistency for task refinements, readability issues (appendix F), and operations that alter the general structure and type of data attributes (e.g. in the sorting example where *university name* is converted from a *nominal* to an *ordinal*).

Appendix F

Enhancing Readability with Graphical & Rendering Transforms

The American Heritage Dictionary defines *readable* as "capable of being read easily". This is a broad definition and can refer to any factor that affects the ease of interpreting a visualization (this corresponds to the *observational* distance of a visualization). In this thesis however, we use *readability* in a narrower sense to refer to

> ***problems arising from constraints of the output media and its interactions with our perceptual system that impede optimal use of a visual design***[1].

F-1 Readability Problems

Different readability problems arise due to our choice of output media. In our work, we are interested in the CRT screen, which has two main restrictions, limited space and limited resolution. Readability problems are also affected by limitations of our perceptual system when interacting with the output media. Three primary limitations of our perceptual system include: 1) Single point of focus (we can only focus on one spatial area at a time), 2) Limited area of focus (our eye is only sensitive to a limited amount of space and is unable to capture visual elements beyond this area of sensitivity), and 3) Limited resolution (our eye can only pick up objects or features at a certain minimum resolution).Based on these limitations we identify four commonly occurring readability problems:

F-1.1 Occlusion

Occlusion occurs when one or more graphical objects in the display visually hinder access to other objects that are important to our task. Occlusion is one of the most commonly recognized readability problem and has been dealt with in many spatial layout algorithms for a variety of visual structures. This is because occlusion problems arise very frequently (especially with the large data sets that we have to deal with today) and cannot be easily avoided. We identify three principal occlusion classes: *line-of-sight occlusion, overlap occlusion,* and *overplotting.*

1. *Line of sight occlusion*: Line of sight occlusion occurs when we try to view a three dimensional scene or a three-dimensional glyph. It is difficult for us to access objects within a three dimensional scene when there are other objects are in front of them in our line of sight because we only have a single point of focus and we cannot see through opaque objects. Line of sight occlusion may occur as a result of: a) self-occlusion or b) inter-object occlusion.

[1] By "*optimal use of a visual design*" we mean the most effective perceptual, cognitive, and articulatory strategy that may be used with a visual design given a non-problematic or "good" data distribution and data set size.

2. *Overlap occlusion*: Overlap occlusion may occur in both three-dimensional and two-dimensional spaces. Unlike line of sight occlusion which occurs for objects along the same line of sight, overlap occlusion occur for objects that are very close to each other spatially. Objects may be partially or fully occluded. It is obvious when and where partial occlusion occurs but for fully occluded objects, we must supply strong cues indicating that there is hidden information. For example, we may provide users with an occlusion map overlay where color in the map represents the amount of occlusion at a spatial region. Another possibility is to move the occluded objects to the front and back automatically. Note that because overlap occlusion is not caused by viewing location but rather by object positions, it cannot be solved by changing the scene viewpoint as was done for *line of sight* occlusion.

3. *Overplotting*: We use overplotting here to refer to the extreme case of overlap occlusion where two objects occupy the exact same position[2]. Unlike overlap occlusion, overplotting cannot be addressed with spatial distortion techniques. Stretching the space between objects (i.e. allocating more space between objects) will not help in this case because the objects are on the exact same location. Thus, the only techniques that can be used in this situation are the object based graphical transform techniques.

F-1.2 Density

When designing visualization displays it is not only important that we avoid occlusion, but also that we reduce visual clutter, i.e. high ink density. High ink density can be distracting to users making it difficult to identify graphical patterns and trends. In general, visual elements should only be introduced if they help facilitate the task so that display density is not unnecessarily increased.

Display density was very carefully studied by Tufte in his book "*The Visual Display of Quantitative Information*" [*Tufte, 1983*]. Tufte presented the idea of *data ink*, which represents the non-erasable core of a graphic. Ink that did not express information required by the task (i.e. *chartjunk*) was wasted ink and only served to clutter up the display. Some example artifacts that commonly result in chartjunk include unintentional optical art (e.g. introducing hash marks and textures that are not needed), overly gridded displays, and art that may beautify the display but does not add to its effectiveness or information content.

Note, however, that removing chartjunk **does not** mean that we should only show the bare minimum amount of data required by the task. If a visual element increases the overall effectiveness of the visual design with respect to the current task(s) then it is **not** chartjunk. Well-rendered grids for example can help direct a user's eye to the positional axes so that value lookups can be performed more accurately. In addition, it may sometimes be advantageous to emboss objects with texture to make them more salient so that we can direct a user's attention to chosen objects in the display. Depending on the task, it may also be

[2] Note that we can reduce the amount of overplotting by using better layout and rendering schemes such as drawing the larger objects behind the smaller objects so that both are visible to the user. However, in those cases where good layout is insufficient to solve the problem, graphical and rendering transforms can be used as well.

effective to multiply encode data attributes (i.e. represent the same data attributes using different graphical properties). Different graphical properties and graphical representations are effective for different data types and tasks. For example, suppose we want to search for houses worth more than 100k as well as be able to accurately look up those sales figures once we find them. For the search task, it is much more effective to encode the values using position because then we can pre-attentively find the houses costing more than 100k. However, text allows for more accurate lookups. By multiply encoding the house *selling-price* attribute as text and bar length, we enable users to perform both the search and lookup tasks effectively.

F-1.3 Dwarfing

Dwarfed encoding scales arise when we have a small range of graphical values representing a large range of data values. As a result different data values may be mapped to similar or non-distinguishable graphical values. Because our perceptual system can only differentiate between graphical values at a certain minimum level of resolution, dwarfed scales often produce highly inaccurate information.

F-1.4 Spatial separation

Our eyes are only sensitive to a limited spatial area. When related data concepts are placed in spatially distant locations, we must not only perform more eye, head, or body movement, but also store information in short term memory from one eye fixation to the next. Thus while it is important to reduce occlusion and display density by separating out the visualization objects, we must balance that with the additional processing required for finding related objects over large spatially distinct locations.

Spatial separation problems arise because of representation constraints (e.g. a bar in a bar chart can only have two other bars next to it), spatial constraints (there is limited adjacent space around a two-dimensional object), and density constraints (areas that are too dense or cluttered are difficult to interpret) that force graphical objects to be spread farther apart.

F-2 Readability Solutions

Readability problems may be addressed in three ways:

1. *Changing the data to graphical mappings within the visualization*: Readability problems can often be avoided by changing the graphical properties and objects used to show the data. However, we must be careful that these changes do not reduce our ability to solve our target tasks (i.e. force a less effective perceptual strategy to be used) or lead to other readability problems.

2. *Changing the balance between data transforms and mapping transforms*: We can also solve readability problems by using data transforms to summarize our data so that fewer data values need to be shown.

Although this method produces visual displays that are less cluttered due to object filtering, the articulatory cost tends to be higher due to the need for user input in directing the filtering process.

3. *Using graphical and rendering transform methods*: Finally, we may also use graphical and rendering transforms to change the appearance of objects in a visualization to solve readability problems in the design. For example, we can allow users to interactively change the transparency of the occluding objects with graphical transform methods. Another possibility is to stretch the display with rendering transforms so that there is more white-space between the objects and less likelihood that they will occlude one another.

Which of the three methods described above is most appropriate for addressing current readability issues depends on how they change the overall *semantic distance*[3] of our visual designs with respect to our task(s). Solution 1 may increase observational distance because less effective data to graphical mappings may need to be used to avoid readability problems. Solution 2 generally increases articulatory load because users need to specify more task parameters when using a highly computed visual design. In addition, expressive distance may also increase because data transform operations often summarize or cull out portions of the source data. Finally, solution 3 adds to articulatory and observational distances because users must interact with the graphical and rendering transform methods as well as interpret their feedback. Nevertheless, solution 3 also allows for more flexibility in picking effective data to graphical mappings.

In chapter IV we considered how readability problems can be avoided by making good mapping and data transform choices (i.e. corresponds to solution 1 and solution 2 above). In this section, we focus on solution 3 and consider how a subset of the graphical and rendering functions described in chapters II and III can be used to solve readability problems. Specifically we consider five basic visualization functions: three graphical and two rendering. These methods were chosen because of their simplicity, and their ability to generate interesting and useful behaviors for addressing readability issues.

In general, rendering techniques differ from graphical techniques because they do not change the graphical objects within a visualization. Instead, they change the way in which these graphical objects are mapped onto an output media. Unlike rendering techniques, graphical techniques are constrained by the *graphical class* of their input objects. A *graphical class* description defines the visual appearance of graphical objects as well as their graphical properties. For example, in our framework, objects belonging to the *horizontal-bar graphical class* has properties *x-length, y-position, color* and *thickness*. Graphical techniques can only operate on these properties and none other. For example, graphical techniques cannot alter the rectangular shape of horizontal-bar objects because the *horizontal-bar graphical class* constrains the objects to be rectangular and do not provide any properties for changing the bars in a non-rectangular

[3] *Semantic distance* measures the goodness of a design with respect to a set of tasks. We described *semantic distance* in detail in chapter 4 and we expand our *semantic distance* model in this chapter, in section 2.

way. Thus graphical techniques guarantee perseverance of "object design" as defined by their *graphical class* (i.e. a horizontal bar will always appear as a horizontal bar before and after the graphical transform method). Rendering techniques on the other hand are not constrained by *graphical class* properties, as a result they may not preserve object appearance. For example rendering methods such as fisheye-lenses and bifocal lenses may distort horizontal-bars so they are no longer rectangular.

F-2.1 Constant Graphical Methods

Constant methods set the values of particular graphical properties to a user or designer declared constant. This is achieved with an *assign* graphical transform that takes a set of graphical property values and a constant as input.

Constant methods are often used to attach a common identifying feature to a set of objects so that users can recognize them as a group. A common application is in providing feedback for search tasks. For example in the *dynamic query slider* technique we search for a set of objects that fulfill certain data attribute constraints by marking that constraint on a slider input device. Objects that fulfill the search constraints will then have their *color* or *transparency* values changed to a common constant for easy perceptual identification. Constant methods can also be used to support occlusion problems by setting the *visibility* or *transparency* property of graphical objects.

F-2.2 Additive Graphical Methods

Additive methods add or subtract a constant to or from a set of graphical values. Additive graphical methods use a *binary graphical transform*[4] with the addition operator (+) as input. As with constant methods, additive methods can be used to provide feedback (e.g. used to change an encoding graphical parameter in a consistent way so that we can identify the objects as a set) or to address readability problems such as removing objects from occlusion or enlarging them so that they can be easily perceived.

The most important difference between additive methods and constant methods is that additive methods maintain the relative ordering among the transformed values while constant methods do not. I.e. if height *a* is less than height *b* before the transform, then this relationship will still hold after an additive transform but not after a constant transform. Therefore, when transforming data encoding graphical properties, it is preferable to use either *additive* or *multiplicative methods* because they both maintain relative ordering. If we use constant methods we lose all of the encoded data information. On the other hand, constant methods are more effective at perceptually grouping a set of objects because they only require users to detect absolute feature similarity while additive and multiplicative methods require users to identify trend similarity (i.e., have the objects changed in a similar way?) which is more difficult to perform.

[4] Binary graphical transforms are described in chapter III-1.2.2.2

F-2.3 Multiplicative Graphical Methods

Multiplicative methods change a set of data values by multiplying them with a constant. Like additive methods we use a *binary graphical transform* but with the multiplication (*) operator as input. Multiplicative methods primarily allow us to stretch or contract the encoding range for the set of chosen graphical values (i.e. representing the same data values with more/fewer graphical values).

As with additive methods, multiplicative methods maintain relative ordering among the transformed values, however, multiplicative methods also maintain ratio relationships. This means that if height *a* is two times greater than height *b* before transformation, it will still be two times greater after a multiplicative transform. Additive methods do not preserve such ratio relationships. For example, once we transform both heights *a* and *b* by adding a constant *c* (where c $\neq$ 0), the ratio relationship between the two values no longer holds true, i.e. $a + c$ does not equal $2(b + c)$. An exception is when we apply additive methods to positional properties, for example, in the *SDM* positional shift operations [*Chuah, 1995*].

F-2.4 Linear Positional Rendering Methods (Point of View Navigation)

Linear rendering functions translate the graphical scene verbatim onto the output media. I.e. the relative position and surface properties of objects in the graphical scene remains the same on the output media. In this appendix section, we only consider rendering methods that are applied to positional properties (e.g. *x-position, y-position*) because they are most effective for solving the readability problems that we are interested in. We leave the treatment of non-spatial rendering techniques (e.g. lighting effects, wire-frame rendering) for future work.

Linear positional rendering techniques (Point of view navigation) allow users to view different sections of a visualization scene. This can be achieved by controlling the camera viewpoint on the scene or controlling the scene itself (i.e. moving or rotating the scene). Both classes of techniques transform desired portions of the graphical scene onto an output media and allow users to explore subsets of the graphical scene without introducing any spatial distortions.

F-2.5 Non-linear Positional Rendering Methods (Distortion)

Non-linear functions distort the graphical scene by changing/distorting the positional and/or surface property relationships among objects. Distortion methods unlike point of view methods do not change the viewpoint on the scene (i.e. the portion of the graphical scene being translated onto the output media remains constant). Instead, distortion techniques stretch certain sections of the graphical space while contracting other surrounding areas. This allows users to focus on particular sections of the visualization while maintaining surrounding context unlike the linear rendering methods described above.

F-3 Applying Graphical and Rendering Methods to Readability Problems

Table F-1 summarizes the expressiveness and effectiveness of the graphical and rendering methods described in section F-2 with respect to the four readability issues (occlusion, density, dwarfing, and spatial separation) discussed in section F-1.

		Occlusion			**Density**	**Dwarfing**	**Spatial separation**
		line of sight	overlap	overplot			
Graphical transforms:							
constant	*positional*	-	-	-	-		+
	spatial retinal	-	-	-	-		
	non-spatial retinal	+*	+*	+*	-*		
additive	*positional*	+	+	+	+		+
	spatial retinal	+	+	+	+		
	non-spatial retinal	+*	+*	+*			
multiplicative	*positional*	-	-	-	-	+	+
	spatial retinal	-	-	-	-	+	
	non-spatial retinal	-*	-*	-*		+	
Rendering transforms:							
Point of view	*positional*	+			+	-	+
Distortion	*positional*	-	-		+	-	+

Table F-1: Expressiveness and effectiveness of graphical and rendering transforms with respect to readability. + : Readability issue is supported reasonable well; - : Readability issue is not supported well; "*empty*" : Readability issue is not supported;* indicates the transparency property only.

Each graphical transform class in Table F-1 is divided into the three classes of graphical properties that may be altered: 1. positional properties (*x-position, y-position*), 2. spatial retinal properties (*orientation, size*), and 3. Non-spatial retinal properties (*color, shape*))[5]. It is important to consider the range of graphical property classes because they affect the effectiveness of the graphical and rendering functions with respect to readability problems. For example, changing the *color* property is effective for drawing a user's attention but it is not too helpful for solving occlusion problems, which require a change in the *visibility* property. Note from Table F-1 that non-spatial retinal properties are not very effective for dealing with most of the readability issues (occlusion, density, and spatial separation) considered here

except for the *transparency* retinal property that can be useful for addressing occlusion problems. Non spatial properties are less effective because the readability issues we address are inherently spatial in nature and cannot be easily resolved by changing non-spatial properties.

F-3.1 Occlusion

The expressiveness of the five graphical and rendering methods discussed in section F-2 with respect to the line of sight occlusion problem is summarized in column 2 of Table F-1. In this case all the techniques can be used to address the line of sight occlusion problem[6] however some methods are more effective (i.e. have a lower *expressive* or *observational distance*) than others. In particular, *constant* methods are not as effective as some of the other methods because it results in the greatest loss of information (i.e. both relative and ratio relationships are lost). Multiplicative methods are also not too effective here because they only allow us to control the positional scale for a set of objects and it is easier to remove occlusion when we have control of the absolute objects positions as is the case with additive transforms. Transparency graphical methods can also be used to address occlusion problems. A possible weakness here is that it is more difficult for us to access information from transparent or translucent objects (higher *observational distance*). However, if the objects we transform are not pertinent to the task then this has no effect on the overall *task semantic distance*. Of the two rendering methods, the *point of view* methods are rated higher here because they allows us to solve line-of-sight occlusion problems with less visual distortion than the *distortion* methods (lower *observational distance*).

From Table F-1 we see that the two rendering techniques, point of view navigation and distortion techniques can be used to solve line of sight occlusion and overlap occlusion. However, these rendering techniques are not expressive of the overplotting problem because they can only change the spatial distance between differently positioned objects, while overplotting is caused by identically positioned objects.

F-3.2 Density

Graphical transforms can be used to reduce ink density by creating *context sensitive displays*. Context sensitive displays allow users to make portions of data ink visible or non-visible depending on the current focus objects. In this way, we may turn on the axis grid lines when we need them for accurate lookups and then turn them off otherwise using a *transparency constant* graphical transform. Graphical transforms can also be used to move ancillary objects away from the focus regions or to minimize the size of those objects to reduce area density. Note that unlike occlusion problems, additive and multiplicative transparency techniques are not expressive of the density problem because simply making a set of objects more or less translucent does not change object density of an area.

---

[5] Note that another possible graphical property class is the temporal (time) dimension. Temporal techniques are not shown here because they are always paired with either a positional or a retinal property, and their expressiveness is dependent on the expressiveness of the paired/linked property.

Apart from the total number of graphical elements within the display, density is also dependent upon the total amount of display space available. Thus another alternative to lowering display density is to render a smaller portion of the graphical scene onto the given output space by dividing the scene into multiple segments and using p*oint of view* rendering techniques to view each of the scene segments separately. However, this lowers the expressiveness of the visualization (not all the information can be shown simultaneously) and may increase the spatial separation among related data concepts. We can also reduce the density of an area by stretching out the area using *distortion* rendering techniques.

F-3.3 Dwarfing

Dwarfing problems may be avoided if we use data transform techniques to summarize the results of our tasks as was shown in chapter IV. Another alternative is to use *multiplicative* graphical transforms on certain ranges of the dwarfed graphical property to expand the graphical value differences among the objects that are of current interest. Constant and additive graphical transform methods are not expressive of the dwarfing problem because they do not have any effect on graphical value encoding scales. When dwarfing occurs on positional graphical properties, we can also address the problem with either of the positional rendering methods. For example, we can divide our data set into several information segments and navigate from one segment to another using point of view navigation. This allows us to encode a smaller data range in each segment, and thus dwarfing is reduced. However, this separation makes it difficult for us to compare values that are in different segments and requires additional articulatory load for navigation. Distortion rendering techniques can also be used to expand the dwarfed areas. However, distortion techniques are less appropriate for the dwarfing problem because they may distort the positional encoding scale, making it difficult to accurately translate the distorted positions back to data values.

F-3.4 Spatial separation

A solution to the spatial separation problem is to use graphical transform methods to move graphical objects to different positions in the display so that the objects that must be compared are never too far from one another. Another alternative is to use rendering distortion techniques to map portions of the graphical scene onto a smaller vertical space, thereby reducing the vertical distance between graphical objects. We can also apply point of view rendering to map the different graphical scene segments we want to consider next to each other. When using these graphical and rendering techniques, however, we must be careful not to overly increase display density or cause occlusion among the elements of interest.

F-4 Graphical and Rendering Transform Guidelines

Design guidelines for selecting mapping transforms (i.e. selecting data to graphical mappings) were set forth in previous work on automatic visualization design. Design guidelines for selecting data transforms and contrasting their use with mapping transforms was discussed in chapter IV. Here we

[6] The only exception is that *positional* methods **cannot** be used to solve self-occlusion problems.

consider guidelines for using the last two transformation classes: graphical and rendering transforms. We focus on rules for selecting effective graphical and rendering methods for solving the four readability issues that commonly arise in visualization displays (occlusion, density, dwarfing, and spatial separation). These design guidelines are aimed at reducing the semantic distance measures described in chapter IV-2.

F-4.1 Relevance of Readability Problems with respect to Tasks

Many readability problems may arise within a visualization design, however, not all of these readability problems are as important or relevant to the current data analysis tasks. Solving all readability problems will require introducing many visualization techniques into the design and this will invariably result in resource conflicts (graphical property and input device conflicts) among the techniques. Thus the first step in choosing appropriate graphical and rendering techniques is to determine which readability problems are most relevant and important. The readability problems that are most important and relevant are those that most affect the end user's ability in solving their goals. Readability issues that do not affect the task or occur over objects that are unrelated to the task need not be addressed. Whether a readability problem significantly affects our task solution depends on the following three factors:

1. *Task specification (task operators and task arguments)*: The relevance of readability problems depend first and foremost on whether the data concepts they affect are important to our current task(s). Both the task operator and the task arguments affect the relevance of readability problems within a visual display. To effectively address readability problems we must first identify which ones are most relevant based on our task and task input arguments.

2. *Distribution of data values*: The relevance of readability problems also depends largely on the distribution of data values that are related to the task. For example finding the *max* value within a dwarfed data set that only has one clear high value is much easier than finding the *max* value within a dwarfed data set with many high almost equal values.

3. *The accuracy required of the task results*: Task accuracy is also an important factor in determining the relevance of readability problems. Readability problems often lower the accuracy of task results, thus depending on the level of accuracy required by the current task, different readability issues may take precedence. Accuracy also affects the effectiveness of graphical and rendering techniques. Some of these techniques distort or change the graphical representations in the visualization display so that it no longer shows the exact data values and relationships in the original data set. Depending on the task accuracy required, these distortions may affect task expressive and observational distance to different degrees.

In summary to design an effective visualization system, we must identify and order existing readability problems based on their importance and relevance to our current data analysis tasks. We then

attend to the readability issues according to their order of importance, assigning better feedback properties and input controls to the problems that have greater significance on the overall semantic distance.

F-4.2 Continuity

Continuity in this context refers to gradual visual transformation. For a change to occur gradually we divide it into multiple smaller changes all occurring within an acceptable time period of the other as to give the appearance of animation or movement. Animated or continuous techniques provide users with better context on how a visualization technique changes the display and which objects are affected (i.e. improves *technique observational semantic distance*). Only continuous graphical properties (e.g. position, saturation, size, length) can be animated. Non continuous graphical properties (e.g. shape, texture) can only be changed discretely. It is however possible, although not desirable, to change continuous properties in a discrete way by making the entire change in a single non-animated step.

An example discrete technique is the painting technique [*Becker, 1987*], which changes the hue of objects discretely so that they appear more salient. Another example is the bifocal lens technique, which commonly changes the size and position of objects in a single visual step that can be quite jarring to users. Some example continuous techniques include the techniques within the SDM system that allows users to manually and continuously change particular graphical properties of objects by selecting and dragging on object handles. Some other examples of continuous techniques include the node rotations within a ConeTree [*Robertson, 1991*], the stretching techniques presented by Sarkar et. al. [*Sarkar, 1993*], or the tree reorganization operation in Hyperbolic trees [*Lamping, 1995*].

Continuous techniques reduce *technique expressive distance* because they provide users with good constant updates of the state and progress of a visualization technique. Continuous feedback of intermediate states also enables users to more easily detect technique errors and bugs. Continuous techniques also reduce *technique observational distance* because they provide users with better context on how a visual representation has changed from its initial state as a result of the visual transformation. In addition they also give users with more time to focus on the interesting objects while a change is occurring. On the other hand, continuous feedback is less useful when the visual changes are small and well understood (i.e. when the observational distance of the design is small to begin with) or if the time addition due to the continuous change is very significant (e.g. when the visual transform is repeated many number of times).

Another related issue is that of spatial continuity. This refers to whether the changes brought about by the graphical and rendering techniques create a spatial discontinuity in the display. The bifocal lens technique [*Leung, 1989*] for example creates a spatial discontinuity because the display is divided into two disjoint sections that are rendered at different levels of magnification. The same effect arises in the magic lens, and table lens techniques. Such spatial discontinuity may be jarring to users as was shown by Hollands et al. in a user test comparing bifocal lenses to fisheye lenses.

F-4.3 Individual vs. Group Readability

As was discussed in appendix C-3, there are two general tasks classes: simple value pair tasks or more complex group tasks. For value-pair tasks we must address readability issues for individual task related objects. Usually, single objects are selected for transformation many times, thus the frequency of executing the readability technique is high. Consequently, we want to keep the cost of each readability operation to a minimum even at the cost of greater initial learning time.

For group tasks we solve readability issues for an entire group of objects (not single objects as was in the previous case). For example when solving group occlusion issues we are more concerned with showing the general shape of the group, (i.e., we are more concerned with occlusion at the group edges) rather than with internal occlusion among individual objects within the group. Unlike value-pair tasks, group tasks require a readability technique to be repeated less frequently because operations are applied to groups rather than individual objects. Thus it is less important that we keep the input load (i.e., *technique semantic distance*) low and more important that we keep *task semantic distance* to a minimum.

F-4.4 Object spatial proximity

Rendering techniques are effective for addressing readability problems that occur among spatially proximal objects because rendering transforms operate on spatially contiguous regions and applying a single transform may remove readability problems from several objects simultaneously. To address the same readability problems using graphical transforms would require a higher articulatory distance because each of the task related object must be enumerated by the user[7]. In addition, more visual feedback must be provided to indicate the location and extent of changes introduced by the graphical transforms. This results in a higher *observational distance*.

However, if the target readability objects were spatially distant from one another, we would need to apply multiple rendering distortions to spatially disjoint areas thereby increasing the feedback complexity and the amount of distortion introduced into the display. Articulatory distance is also higher because now users must specify multiple focal points. On the other hand, graphical transforms become more effective in this case because they introduce less visual distortion compared to rendering transforms that are applied over multiple separate focal points.

F-4.5 Reversibility

Reversibility refers to how easily the changes made by a graphical or rendering transform may be removed from a visual display. Reversibility of graphical and rendering transforms is important for three reasons:

[7] The articulatory load may be reduced by using functional definition selections, (e.g. using a bounding box to set constraints on the position of the objects desired).

1. *Reduces conflict among different transformation techniques*: Some graphical and rendering techniques may introduce new readability problems into the visual display. For example, the fisheye technique increases density of the contextual areas and distorts the position of objects within the display. As a result, we may want to remove the effects of the fisheye transformation once we perform our immediate task so that the display distortions it introduced will not influence other tasks that need to be performed. The ability to reverse the transformation allows us to deal with the readability issues on a task by task basis and not have to worry about conflicts in readability operations across different tasks. This reduces the task expressive and observational distances of a visualization technique.

2. *Enables users to easily repeat visual changes that were missed*: Reversible transformations are also useful for repeating a graphical or rendering transform. Sometimes, we may not catch all the visual changes made by a technique. Being able to reverse and reapply the technique gives us the ability to peruse the effects of the technique repeatedly and more effectively capture the visual alterations that have occurred. I.e. this lowers technique observational distance.

3. *Reduces the cost of input and operation sequence errors*: Interactive visualization techniques allow users to control or manipulate part of its operation and effects through input devices. Sometimes however, users may make input errors and it is important to allow the effects of those error(s) to be reversed. User errors may also occur when we need to perform a series of graphical and rendering operations in sequence to get combined effects on a set of objects. In such an instance if we accidentally perform an operation out of sequence, we must be able to reverse it so that we need not repeat the entire sequence again from scratch. Reversible techniques reduce the articulatory cost of making input errors or operation sequence errors.

F-4.6 Learning

The effectiveness and usability of a technique is increased if it can be easily learned and if it is easy to use and remember once learnt. Some important learning issues include:

1. *Consistency* : Techniques within the same visualization should share consistent interfaces and controls. This helps users remember the techniques and enables them to transfer knowledge from one technique to another. To ensure technique consistency we can decompose the techniques based on the interactive framework presented in chapters II and III, and then give preference to techniques with similar structures.
2. *Vocabulary size*: The more techniques there are in a visualization system (i.e. the greater the technique vocabulary size), the harder it is for users to master and utilize these disparate techniques. Thus we should try to keep the number of controls and number of different techniques to a minimum.
3. *Affordances*: This refers to whether affordances or cues are provided to users for indicating how a visualization technique should be used and manipulated and what problems it can address. Such cues can take many forms. We can provide instructional support by integrating the automatic visualization

designer with an explanation system like AutoBrief, which can provide textual instructions. Effective affordances can also be derived by picking input devices whose appearance suggests the inputs it can generate and the way it should be manipulated.

Note that the three factors presented here are obviously not a complete list of factors that affect learning a technique. Because of the breadth and complexity of learning issues however, we leave a more complete treatment of it for future work.

F-5 Conclusion

In this section we identified four important readability issues (*occlusion, density, dwarfing* and *spatial separation*) by examining constraints of the CRT screen and out perceptual system. We then considered how the graphical and rendering transform techniques captured within our framework in chapter III can be used to address these four problems.

To add these readability decisions into our automatic design system, we must add several additional steps in our search algorithm as is shown in gray in Figure F-1. In particular, once we have finished constructing a design, we check to see if that design has any readability problems. If so, we consider all available graphical and rendering techniques and pick one that is most appropriate given the current task, data set size, data value distribution, and according to the guidelines and metrics discussed in chapter IV-2 and section F-4.

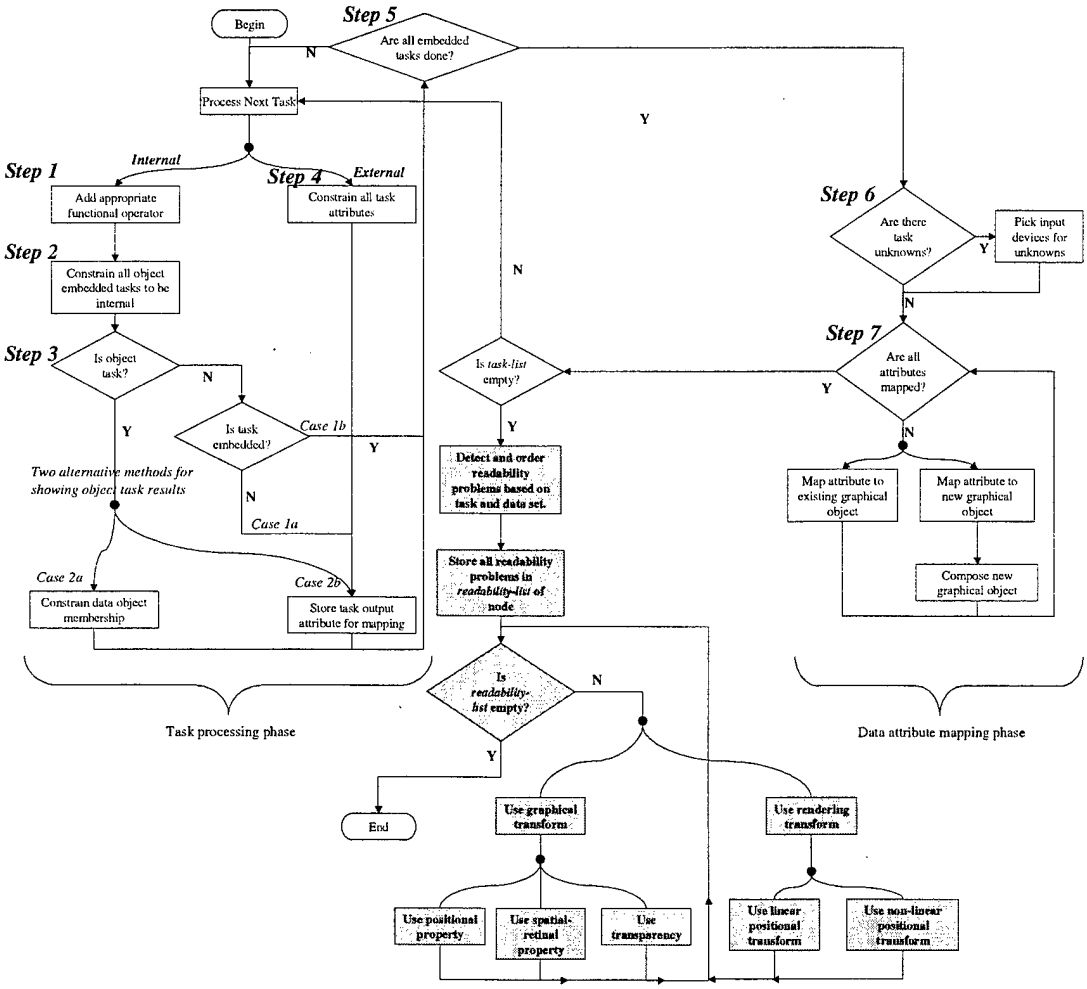


Figure F-1: Augmented search algorithm for our automatic design system AVID. Additional steps take into account readability issues and how to solve them.

References

Abowd, G., and Beale, R., (1991). Users, Systems and Interfaces: A Unifying Framework for Interaction. *People and Computers VI, Proceedings of the HCI'91 conference,* editors Diaper D., and Hammond N., p.73–87.

Ahlberg, C., Williamson, C. and Shneiderman, B., (1992). Dynamic queries for information exploration: An implementation and evaluation. *Proceedings of CHI'92 Human Factors in Computing Systems*, ACM, Monterey, CA, p. 619-626.

Ahlberg, C., Shneiderman, B., (1994). The Alphaslider: A Compact and Rapid Selector. *Proceedings of CHI'94 Human Factors in Computing Systems*, ACM, p. 365-371.

Becker, A., and Cleveland, W. S., (1987). Brushing Scatterplots. *Technometrics,* vol. 29, no. 2, p. 127-142.

Bederson, B.B., and Hollan, J.D., (1994). PAD++: A zooming graphical interface for exploring alternate interface physics. *UIST '94, Proceedings of the ACM Symposium on User Interface Software and Technology*, ACM, p. 17-27.

Bertin, J., (1983). *Semiology of Graphics*, The University of Wisconsin Press, London, England.

Bertin, J., (1981). *Graphics and Graphic Information Processing*, Walter de Gruyter, Berlin, New York.

Bier, E. A., Stone, M.C., Baudel, T., Buxton, W., and Fishkin, K., (1994). A Taxonomy of See-Through Tools, *Proceedings of CHI'94 Human Factors in Computing Systems*, ACM, Boston, MA, p. 358-364.

Brodlie, K. W., Gallop, J. R., Grant, A. J., Haswell, J., Hewitt, W. T., Larkin, S., Lilley, C. C., Morphet, H., Townend, A., Wood, J., Wright, H. (1991). Evaluation of Visualization Software, *AGOCG Technical Report 9.*

Card, S.K., Mackinlay, J., Shneiderman, (1999). *Readings in Information Visualization: Using Vision to Think*, Card, S.K., Mackinlay, J., Shneiderman (Eds), Morgan Kaufman Publishers Inc., San Francisco.

Card, S.K., Mackinlay, J., (1997). The Structure of the Information Visualization Design Space, *Proceedings of the Symposium on Information Visualization,* IEEE, Phoenix, AZ, pp. 92-99.

Card, S.K., Mackinlay, J., Robertson, G., (1990). The Design Space of Input Devices, *Proceedings of CHI'90 Human Factors in Computing Systems*, ACM, p. 117-124

Card, S.K., Moran, T.P., and Newell A., (1983). *The Psychology of Human-Computer Interaction,* Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Casner, S.M., (1991). A Task-Analytic Approach to the Automated Design of Graphic Presentations, *Transactions on Graphics*, ACM, Vol. 10, No. 2, p. 111-151.

Cavanaugh, J.P., (1972). Relation between the intermediate memory span and the memory search rate, *Psychological Review,* Vol 79, p. 525-530.

Chernoff, H., (1973). The Use of Faces to Represent Points in k-Dimensional Space Graphically, *Journal of the American Statistical Association*, p. 361-368.

Chuah, M., Eick, S.G., (1998a). Information Rich Glyphs for Software Management Data, *IEEE Computer Graphics and Applications*, IEEE, p. 24-29.

Chuah, M., Roth, S.F., (1998b). Dynamic Aggregation with Circular Visual Designs, *Proceedings of the Symposium on Information Visualization*, IEEE, North Carolina, p. 35-43.

Chuah, M., Roth, S.F., (1996). On the Semantics of Interactive Visualizations, *Proceedings of the Symposium on Information Visualization*, IEEE, San Francisco, CA, p. 29-36.

Chuah, M., Roth, S.F., Mattis, J., Kolojejchick, J., and Juarez, O., (1995). SageBook: Searching data graphics by content, *Proceedings of CHI'95 Human Factors in Computing Systems,* Denver, CO, p. 338-345.

Chuah, M.C., Roth, S.F., Mattis,J., and Kolojejchick, J., (1994). SDM: Selective Dynamic Manipulation of Visualizations, *UIST'95 Proceedings of the ACM Symposium on User Interface Software and Technology*, ACM, Pittsburgh, PA, p. 61-70.

Eick, S. G., Steffen, J. L., Sumner, E., (1992). Seesoft – A Tool for Visualizing Line Oriented Software Statistics, *IEEE Transactions on Software Engineering,* Vol 18, IEEE, p. 957-968.

Feiner, S., and Beshers, C., (1990). Worlds within worlds: Metaphors for exploring n-dimensional virtual worlds, *UIST'90 Proceedings of the ACM Symposium on User Interface Software and Technology*, ACM, p. 76-83.

Foley, J.D., vanDam, A., Feiner, S.K., and Hughes, J.F., (1990). *Computer Graphics: Principles and Practice,* Addison Wesley Publishing Company.

Furnas, G.W., (1991). Generalized fisheye views. *Proceedings of CHI `91 Human Factors in Computing Systems*, ACM, p. 16-23.

Goldstein, J., Roth, S.F., Kolojejchick, J., and Mattis, J., (1994). A framework for knowledge-based interactive data exploration, *Journal of Visual Languages and Computing*, p. 339-363.

Ellson, J., Gansner, E., Koutsofios, E., Mocenigo, J., North, S., Woodhull, G., Dobkin, D., Alexiev, V., GraphViz, http://www.research.att.com/sw/tools/graphviz/.

Green, M., Visual search, visual stream, and visual architectures, *Perception and Psychophysics,* in press.

Hollands, J.G., Carey, T.T., Matthews, M.L., and McCann, C.A., (1989). Presenting a Graphical Network: A Comparison of Performance Using Fisheye and Scrolling Views, *Designing and Using Human-Computer Interfaces and Knowledge Based Systems,* Elsevier Science B.V., Amsterdam, p. 313-320.

Jacob, R., (1986). A Specification Language for Direct Manipulation User Interfaces, *Transactions on Graphics,* Vol. 5, No. 4, ACM, p. 283-317.

John B.E., (1988). *Contributions to engineering models of human-computer interaction.* Doctoral dissertation, Carnegie Mellon University.

John, B.E. & Newell, A., (1989). Cumulating the science of HCI: From S-R compatibility to transcription typing. *Proceedings of CHI'89 Human Factors in Computing Systems,* Austin, Texas, ACM, p 109-114.

John, B., (1990). Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information. *Proceedings of CHI'90 Human Factors in Computing Systems*, Seattle, WA, ACM, p. 107-115.

John, B.E., & Newell, A., (1990). Toward an engineering model of stimulus response compatibility, *Stimulus-response compatibility: An integrated approach*, R.W. Gilmore & T.G. Reeve (Eds), New York:North-Holland, p. 107-115.

Kerpedjiev, S., Carenini, G., Roth, S. F., Moore, J. D., (1997). AutoBrief: a multimedia presentation system for assisting data analysis, *Computer Standards and Interfaces*, Volume 18, p. 583-593.

Lamping, J., Rao, R., Pirolli, P., (1995). A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. *Proceedings of CHI '95 Human Factors in Computing Systems,* ACM, p. 401-408.

Leung, Y.K., (1989). Human-Computer Interface Techniques for Map Based Diagrams, *Designing and Using Human-Computer Interfaces and Knowledge-Based Systems*, Elsevier Science B.v., Amsterdam, p. 361-368.

Leung, Y.K., and Apperley, M. D., (1994). A review and Taxonomy of Distortion-Orientation Presentation Techniques, *Transactions of Computer-Human Interaction*, ACM, Vol. 1, No. 2, p. 126-160.

Livingstone, Margaret S., (1988). Art, Illusion and the Visual System, *Scientific American*, Vol. 256, pp. 78-85.

Lohse, G.L., (1993). A Cognitive Model for Understanding Graphical Perception, *Human Computer Interaction,* Vol 8, p. 353-388.

Lohse, G. L., Biolsi, K., Walker, N., and Reuter, H.H., (1994). A Classification of Visual Representations, *Communications of the ACM*, Vol. 37, No. 12, p. 36-49.

Mackinlay, J.D., Card, S.K., Robertson, G.G., (1991). The Perspective Wall: Detail and Cotext Smoothly Integrated. *Proceedings of CHI'91 Human Factors in Computing Systems*, ACM, New York, 173-180.

Mackinlay, J.D., Card, S.K., Robertson, G.G., (1990). A Semantic Analysis of the Design Space of Input Devices, *Human Computer Interaction,* Vol. 5, Lawrence Erlbaum Associates, Inc., 145-190.

Mackinlay, J.D., (1986a). *Automatic Design of Graphical Presentations,* Ph.D. Thesis, Stanford University.

Mackinlay, J.D., (1986b). Automating the design of graphical presentations of relational information, *Transactions on Graphics*, ACM, Vol. 5, No. 2, pp. 110-141.

Myers, B.A., (1991). Using AI Techniques to Create User Interfaces by Example, *Intelligent User Interfaces*, Sullivan, J.W. (Ed), Reading, MA: Addison-Wesley/ACM Press, p. 385-401.

Olson, J.R., & Olson, G.M., (1990). The growth of cognitive modeling in human-computer interaction since GOMS, *Human-Computer Interaction,* Vol 5, p. 221-265.

Plaisant, C., Milash, B., Rose, A., Widoff, S., and Shneiderman, B., (1996). LifeLines: Visualizing Personal Histories, *Proceedings of CHI'96 Human Factors in Computing Systems*, ACM, Vancouver, BC, Canada, p.221-227.

Rao, R., Card, S.K., (1994). The table lens: Merging graphical and symbolic representations in an interactive focus+context visualization for tabular information, *Proceedings of CHI '94 Human Factors in Computing Systems,* ACM, p. 318-322.

Robertson, G., Mackinlay J.D., Card, S.K., (1991). Cone Trees: Animated 3D Visualizations of Hierarchical Information. *Proceedings of CHI '91 Human Factors in Computing Systems,* ACM, p. 189-194.

Roth, S., Lucas, P., Senm, J., Gomberg, C.C., Burks, M.B., Stroffolino, P.J., Kolojejchick, J.A., (1996). Visage: A user interface environment for exploring information, *Proceedings of the Symposium on Information Visualization,* IEEE.

Roth, S.F., Kolojejchick J., Mattis J., Goldstein J., (1994). Interactive Graphic Design Using Automatic Presentation Knowledge, *Proceedings of CHI'94 Human Factors in Computing Systems,* ACM, Boston, p. 112-117.

Roth, S.F., Mattis, J.A., (1990). Data Characterization for Intelligent Graphics Presentation, *Proceedings of CHI'90 Human Factors in Computing Systems,* ACM, Seattle, WA, p. 193-200.

Sarkar, M., Snibbe, S.S. (1993). Stretching the rubber sheet: A metaphor for viewing large layouts on small screens. *UIST '93 Proceedings of the ACM Symposium on User Interface Software and Technology,* p. 81-91.

Senay, H., Ignatius, E., (1994). A Knowledge Based System for Visualization Design, *IEEE Computer Graphics and Applications*, p. 36-47.

Siskind, J.M., Screaming Yellow Zonkers, http://www.csd.abdn.ac.uk/~swhite/zonkers.html

Spoerri, A., (1993). InfoCrystal: A Visual Tool for Information Retrieval, *Proceedings of IEEE Visualization'93 Conference,* Los Alamitos, CA., p.150-157.

Springmeyer, R.R., (1992). *Designing for Scientific Data Analysis: From Practice to Prototype*, Ph.D. Thesis, Lawrence Livermore National Laboratory.

Teghtsoonian, J., (1965). The judgement of size, *American Journal of Psychology*, p. 392-402.

Treisman, A., Schmidt, H., (1982). Illusory Conjunctions in the Perception of Objects, *Cognitive Psychology,* Vol 14, p. 107-141.

Treisman, A., (1988). Features and Objects: The Fourteenth Bartlett Memorial Lecture, *The Quarterly Journal of Experimental Psychology*, Lawrence Erlbaum Associates Ltd, The Distribution Center, Blackrose Road, Letchworth Herts SG6 IHN, U.K., p. 201-237.

Tufte, E. R., (1983). *The Visual Display of Quantitative Information,* Graphic Press, Cheshire, CT.

Tukey, J.W., (1977). *Exploratory Data Analysis*, Addison-Wesley Publication Co.

Tweedie, L., (1997). Characterizing Interactive Externalizations, *Proceedings of CHI'97 Conference on Human Factors in Computing Systems*, Atlanta, p. 375-382.

Tweedie, L., Spence, R., Williams, D., Bhogal, R., (1994). The Attribute Explorer, *Conference Companion of CHI'94 Conference on Human Factors in Computing Systems,* p. 435-436.

Welford, A.T., (1973). Attention, strategy, and reaction time: A tentative metric, *Attention and performance IV,* S.Kornblum (Ed.), New York: Academic, p 37-54.

Wills, G., (1996). 524,288 Ways to Say "This is Interesting", *Proceedings of the Symposium on Information Visualization,* IEEE.