AFRL-IF-RS-TR-2004-247
Final Technical Report
September 2004

LINK ANALYSIS WORKBENCH

SRI International

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-247 has been reviewed and is approved for publication

APPROVED: /s/
ROBERT L. HAWKINS
Project Engineer

FOR THE DIRECTOR: /s/
JOSEPH CAMERA, Chief
Information & Intelligence Exploitation Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 2004	**FINAL** **Sep 01 – Mar 04**

4. TITLE AND SUBTITLE

LINK ANALYSIS WORKBENCH

5. FUNDING NUMBERS

C - F30602-01-C-0193
PE - 62301E
PR - EELD
TA - 01
WU - 16

6. AUTHOR(S)

Pauline M. Berry, Ian Harrison, John D. Lowrance, Andres C. Rodriguez, Enrique H. Ruspini, Jerome M. Thomere, Michael J. Wolverton

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

SRI International
333 Ravenswood Avenue
Menlo Park CA 94025

8. PERFORMING ORGANIZATION REPORT NUMBER

N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL/IFEA
525 Brooks Road
Rome NY 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-IF-RS-TR-2004-247

11. SUPPLEMENTARY NOTES

AFRL Project Engineer: Robert L. Hawkins/IFEA/(315) 330-2243 Robert. Hawkins@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

In the wake of the 9/11 terrorist attacks on the United States, many questions have been asked and investigations were conducted to determine how these attacks could have happened. The consensus is information was available to warn U.S. intelligence organizations of the impending attacks but were not able to "connect the dots", that is bring the vital pieces of information together to paint a clear picture of what is about to happen. An important role of the intelligence organizations is to identify and track situations of interest – terrorist and criminal activity, signs of impending political upheaval abroad. The problem is the analyst is overwhelmed with intelligence reports from many sources that may provide them with incomplete or misleading information. There is a need for information technology to help the analyst with identification of threat cases, tracking and analysis of the data to confirm the emerging threat. The approach taken here is one of link discovery by matching transactional data stored in a relational database to a well-defined scenario describing a threat situation. The Link Analysis Workbench (LAW) is designed to find patterns (scenarios) buried within vast quantities of large and noisy datasets helping analysts find evidence to support the threat described in the pattern.

14. SUBJECT TERMS

Link analysis, pattern matching, graphs, graph-edit distance, ontologies, relational data, situational awareness, situational assessment

15. NUMBER OF PAGES 87

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

1 Introduction

SRI International (SRI) is pleased to submit this final report to the Defense Advanced Research Projects Agency (DARPA) on SRI Project 11590, "The Link Analysis Workbench."

Immediately after the 9/11 attacks, questions arose regarding why the the U.S. government had been unable to "connect the dots" that would have provided warning of the impending attacks. An important role of intelligence organizations is to identify and track situations of interest—terrorist and other criminal activity, signs of impending political upheaval abroad, and so on—in a sea of noisy and incomplete information. This requires that they find and understand the significance of links between new and previously acquired information. The amount of such information available to these agencies far exceeds the human capacity to analyze it. Therefore, information technology is needed to assist these analysts if they are to succeed.

This situation is not unique to federal intelligence agencies. Essentially the same technical problem confronts forensic accountants, insurance fraud investigators, bank examiners, criminal investigators, computer security analysts, market researchers, medical researchers, and others. The common need is to find patterns within vast quantities of data that reveals what has happened or is about to happen. A large amount of such data is in relational form (i.e., stored in relational databases), and we can expect the amount to increase dramatically in the near future. There is a critical need for technology that helps analysts find and elaborate evidence in relational data.

There are at least three different technical approaches to this problem: (1) find matches in the data for known patterns of interest, (2) find anomalies where known patterns are violated in the data, and (3) discover new patterns of interest. Although the first of these might not seem all that difficult, it is difficult when the amount of data is vast, when the data contains errors of omission and commission because of faulty collection or handling, and when entities are actively attempting to deceive and/or conceal their actions. For example, uncovering known forms of money laundering might be addressed by this method. The second of these is based on the idea that by identifying those things that do not fit the norm, we can discover the true nature of things, despite attempts to conceal them. In general, attempts to do this are exacerbated by the same issues that thwart finding matches from known patterns. An example where these techniques might be successfully applied is a criminal investigation where unusually large purchases or holdings by an individual might suggest a payoff for having engaged in criminal activities.

The third approach, discovering new patterns, can take at least two forms. Social network analysis analyzes the types and frequencies of interactions among people to discern which are acting as groups in pursuit of common goals. Data mining seeks to discover new patterns of indicators that identify events of interest when they cannot be directly observed. These techniques might be used to uncover clandestine organizations or new forms of insurance fraud.

Our approach is based upon finding matches for known patterns of interest. Although some of these patterns might have been defined through the use of data mining techniques, we assume that the vast majority will have been directly defined by analysts, that is, subject matter experts in the field of interest. Specifically, we want to develop tools that help analysts define and match patterns in relational data, where the notion of "match" is very broad and gives the analyst a large amount of flexibility. By broad we mean to encompass both exact and close matches, where close matches might include incomplete matches where some components of the pattern could not be matched, and inexact matches where the precise type of component specified by the pattern was not found, but a semantically similar component was. A key aspect of this approach is that a numeric quality of match is calculated, based upon the absences of and substitutions for components specified in the pattern, indicating the closeness of a match.

For most domains of interest, defining good patterns is difficult. The objective is to define patterns that identify all examples of a target activity without misidentifying any examples. To this end, we endeavor to support the analyst in a generate, test, and refinement cycle, where patterns are defined and matched against the data, the results are analyzed, and modifications are made to the pattern in hopes of improving its accuracy and precision. We expect this cycle to be repeated frequently in the early phases of a pattern's development. Once it matures, we expect the cycle to slow, but typically not stop, since the actions and methods employed by those being analyzed will evolve over time. Thus, we expect a continuous cycle of pattern refinement and spawning of variants. As the actions and methods employed by those being analyzed evolve, we anticipate that the broad notion of match that we support will aid the analysts in tracking these changes. Supporting close matches reduces brittleness; as a method evolves we anticipate that the quality of match for a given pattern will tend to erode slowly, giving the analysts an opportunity to spot these variations by examining the lesser matches. Once spotted, variants on the original pattern can be developed to raise the system's ability to find high quality matches for these new methods. As such, this technique attempts to exploit the identification of anomalies with respect to the original pattern.

Our approach is semi-automated, where analysts remain in the loop, constantly looking to improve the patterns being used. This helps to make our approach agile in the face of evolving threats and/or opportunities. It is our hope that analysts will leverage patterns developed by others. This can take several forms. One form that we support is the hierarchical specification of a pattern that makes use of one or more patterns as subpatterns. As such, higher-level patterns can be developed by assembling lower-level patterns. We also support the use of disjunctive components within a pattern. These are used when there are multiple ways for a given aspect of a pattern to be manifest. It is our intent to foster communities of analysts sharing patterns and results, with appropriate access control, to allow them to leverage each other's successes and failures.

For our approach to succeed, it is essential that the patterns and matches be represented in a form that is intuitive to the analysts, yet rigorous enough to support automated searches for matches. Since analysts performing link analysis often explain their patterns and results through graphical drawings, we adopted a graphical representation. Entities are captured as nodes and relationships as directed connections. These graphical structures are developed by analysts through a tool that allows them to directly "draw" these structures. Matches are depicted through use of the same graphical structures that represent patterns, color coded to indicate the quality of match achieved for each aspect of the pattern. Using graphs in this way eases communication between humans and machines.

If communities of analysts are to work collaboratively in the development of patterns and the exploitation of pattern matching results, it is essential that all potential participants have ready access to the system that we are building to support this. To this end, our approach was to architect the Link Analysis Workbench (LAW) as a Web server with lightweight browser clients. The only requirement for the client machines is that they have an industry-standard Web browser installed and are on a common network with the LAW server. Since Web browsers are now ubiquitous across all personal computers, no matter the specific choice of hardware or operating system, no additional software needs to be installed to give an analyst access to LAW. The LAW system administrator must provide the analyst with only a user name and password to gain access.

While we believe that our approach has the potential to significantly aid analysts in finding patterns within relational data, we do not believe that our approach should be pursued at the exclusion of others. Instead, we imagine that our approach should be used in concert with other techniques based on anomaly detection, social network analysis, data mining, and other link analysis techniques, to collectively pursue the difficult task of identifying and tracking situations of

interest within the changing seas of noisy and incomplete relational data.

This report describes the Link Analysis Workbench (LAW) [49, 45], a system designed to help meet these needs. LAW is designed to allow a domain expert user to build and refine patterns quickly, to search for matches from a large data set, and to understand and compare the matches it finds easily. Because the intelligence domain involves missing data and even imprecise user understanding of what the right pattern should be, LAW is specifically focused on *approximate* pattern matches—situations in the data that are similar to, but not exactly the same as, the situation represented in the pattern. LAW is also designed as a user-centric system, where the pattern representation and matching criteria are understandable by an intelligence expert user, and where the user can play an important, hands-on role in the system's cycle of authoring, matching, and revising patterns.

We begin by describing LAW's pattern representation and pattern matching approach. Then we give an example of LAW's pattern matching behavior and describe LAW's user interface. After that, we describe the architecture and common pattern language on which LAW is based. Next, we discuss our experience with integrating LAW with other link analysis tools in Technology Integration Experiments (TIEs). Finally, we describe some issues explored in our work on an intelligence control component for LAW.

2 Patterns and Matching

The goal of the LAW pattern matching component is to help intelligence analysts[1] find instances in the world of generic scenarios comprising combinations of events and facts about players involved in those events. This problem requires more than a simple database querying capability because of several sources of ambiguity and uncertainty within the process. There may be noise or missing information within the data. The same or similar situations may be represented in multiple different ways. And, most important, the analyst may be able to describe the situation of interest only at a high level or with a limited amount of precision.

The model of patterns LAW uses for this problem of approximate matching is to have each pattern represent two things: (1) a *prototype* of the situation of interest, and (2) allowable *deviations* from the prototype, along with the impact these deviations have on the assessed quality of the match. To fill this model and

[1]Here and throughout the remainder of this report, we use the term "analyst" as shorthand to describe the intended end-user of LAW. In practice, LAW is more broadly intended for any intelligence professional, from low-level analysts to high-level managers in the intelligence community.

to meet the additional requirement of understandability, we have chosen a pattern representation based on graphs, and a match metric based on *graph edit distance*.

2.1 Pattern Representation

A LAW pattern consists of two parts: a graph, representing the prototype situation described above, and a collection of edit distance parameters, representing the allowable deviations described above. We describe the former here, and describe the latter below in Section 2.2.

The graph portion of the pattern representation (called the *pattern graph*) is a collection of typed vertices (nodes), and a collection of labeled edges (links) relating the vertices. Each node in the graph is a *generic concept* [43] of a type, or a literal. The types are organized in an ontology; to access the ontology in LAW, we are using OCELOT [30], an OKBC-compliant [10] knowledge representation system. Labels on edges are also typed with the types organized in the ontology. Specific instances can be approximated in the pattern using literals. For example, the person Michael Wolverton can be represented by attaching a PERSON node to a node containing the string "Michael Wolverton" via a NAME relation.[2]

Our design goal for the pattern graph representation is to give to the analyst a representational capability that is powerful, but still understandable to a lay-user and reasonably efficient to match. In particular, we want a pattern language that stops well short of the representational power and inferential capabilities of first-order logic or conceptual graphs [43], but still goes beyond the capabilities of simple flat typed graphs. Toward this end, we extended the design of the pattern graph representation to include notions of hierarchy, disjunction, and cardinality.[3] For example, Figure 1 shows a hierarchical pattern representing a murder-for-hire. The pattern hierarchically includes two subpatterns, one of which itself includes a nested subpattern. In this figure, the circles represent *interface nodes*, nodes that connect a subpattern to its parent pattern. We upgraded the pattern comparison metric and the matching algorithm to handle the more advanced pattern graph representation, and extended LAW's pattern matcher to match graphs hierarchically.

[2]This encoding represents *any* person named Michael Wolverton, which is often sufficient for the purposes of pattern matching. If the user needs to identify the person of interest more narrowly, additional qualifiers (e.g., birthdate) can be used.

[3]By cardinality, we mean specifying information about the number of links, nodes, or subgraphs, e.g. "three or more meetings."

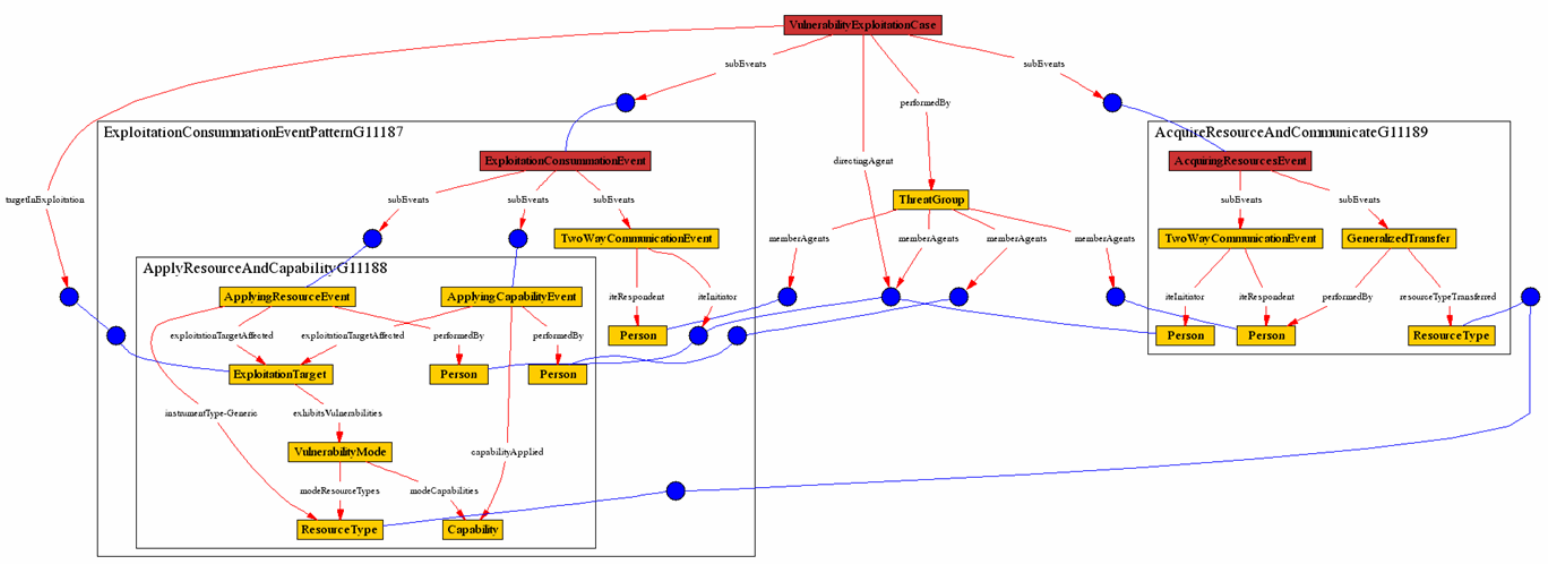


Figure 1: Hierarchical pattern from EELD Y2 challenge problem domain

2.2 Pattern Comparison Metric

The term "graph edit distance" covers a class of metrics that measure the degree of match between two graphs. Variants have been studied theoretically [8, 9] as well as applied in domains as diverse as image understanding [41] and reasoning by analogy [48]. In its simplest form, the graph edit distance between two labeled graphs G_1 and G_2 is the smallest number of editing operations it would take to transform G_1 into G_2. Allowable editing operations are node addition, node deletion, edge addition, edge deletion, node label replacement (i.e., changing the label attached to a node from one term to another), and edge label replacement. This simple model can be extended by adding costs to the various editing operations, perhaps as a function of the labels on nodes or edges, and measuring the edit distance between two graphs as the minimum cost of a sequence of operations that converts one into the other.

LAW uses the more complex model of associating costs with operations. The current LAW model uses only three of the six aforementioned editing operations: node deletion, edge deletion, and node replacement.[4] Each node and edge in a

[4]Node addition and edge addition are not relevant in pattern matching (unlike, for example,

LAW pattern graph has an associated cost for deleting it (see below). For node label replacement, LAW uses the ontological distance between the types of the pattern node and the mapped data node.

The collection of edit distance parameters contained within a LAW pattern specify the allowable deviations from the prototype that will still be considered a valid match, and the cost that various deviations have on the overall quality of the match. These parameters control the calculation of the edit distance between the pattern and the data. They include

- a *deletion cost* on each node and link in the pattern. Each of these can be a number, which roughly reflects the node's or link's level of importance in the pattern. They can also be set to a symbol representing infinite cost, which indicates that the node or link must be matched by a node or link in the data.

- a *maximum ontological distance* on each node in the pattern. This specifies the allowable distance between the type of a pattern node and the type of a node in the data that matches it. Setting this to 0 indicates that the pattern node must be matched exactly, e.g,, a PHONE-CALL node in the pattern can only match a PHONE-CALL node in the data. Setting it to a number greater than 0 indicates that the node can be matched to a node of another type, e.g., a PHONE-CALL node in the pattern can be matched to other subtypes of COMMUNICATION.

- an *ontological distance multiplier*, which specifies the cost of mapping a node of a given type in the pattern to a node of another type in the data. This factor specifies how much penalty the match will pay, for example, for matching a PHONE-CALL node in the pattern to an EMAIL node in the data.

- a *maximum total edit distance* for allowable matches. No matches that are above this threshold will be returned, and any partial matches that exceed this threshold will be pruned from the system's search.

- the *maximum number of matches* for the system to return.

analogical reasoning), because of the asymmetry between pattern and data: we are not trying to make the pattern look like the entire data set, only a small portion of it. And while edge replacement could be a useful construct in pattern matching, we have not yet found a need for it in practice in our use of the system.

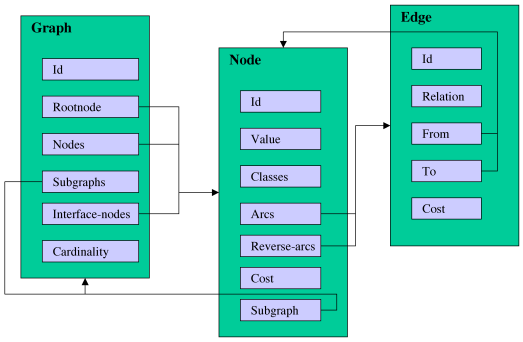


Figure 2: GEM data structure

2.3 Matcher Implementation

2.3.1 Pattern Representation Language: GEM

To represent patterns and data to be used by the matcher, we use a direct graph representation: GEM (Graph-Edit Model). Figure 2 graphically depicts the structure of the GEM language.

The graph (Graph) is a collection of typed vertices (Node), and a collection of labeled edges (Edge) relating the vertices. Each node in the graph is an instance of a class in the ontology, or a value (e.g., a string or a number). Labels on edges are also typed with the slots present in the ontology.

The main intent behind that representation is efficiency. It is achieved specifically by the redundancy between linking nodes and edges together. Similarly, the "Nodes" field of a graph is implemented with a hash table.

The API to access the GEM structures allows basic operations, such as adding and removing nodes and edges. It also permits efficient access from a node to all the edges pointing to it or coming out it, and efficient access from an edge to its adjacent nodes.

In addition, GEM can represent hierarchical graphs with the fields "Subgraphs" and "Interface-nodes" of the Graph element. Figure 3 is an example of the use of GEM to represent subgraphs.

Relations to other LAW representations The GEM representation is intimately linked with other representations that exist in LAW, not only PatternML but also the ontological representation and EDB.

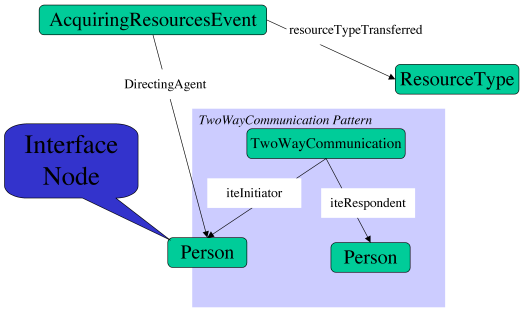


Figure 3: Example of hierarchical patterns

With respect to the ontology, nodes correspond to instances of classes. It is worth noting that each node has the complete hierarchy of classes of the instances represented in the "Classes" field of the node. Edges represent slots; more precisely each edge represents one particular slot value of an instance. Since we are representing relationships, most of the values are instances themselves, but they can also be numerical values, or strings. In these latter cases, they are still represented by nodes in GEM.

With respect to the EDB schema, basically, entities are represented as nodes and links as edges. There are few attributes in the EDB schema. They are represented as edges, and their values are represented as nodes of the class "Value." It allows a good homogeneity of the structures that compose the graph. One drawback of the approach is that it can lead to huge graphs that would be difficult to manage.

Data representation Since the principal aim of GEM is to match pattern and data, the representation of data is almost identical.

The representation of data is also based on GEM. The main addition is that the "Value" field will always be filled by the instance of the node in the data.

2.3.2 Matching Algorithm

LAW's current approach to finding the closest matches to the pattern in the data is based on A* search [22]. A state in the search is a partial match—a mapping between a subset of the pattern nodes and data nodes, a mapping between a subset of the pattern links and data links, a set of unmapped nodes, a set of unmapped links, and a cost of the mappings so far. The cost is the sum of the delete costs of the

unmapped nodes and link, and replacement cost of the node and link mappings, as described above.

LAW generates start states for the search by selecting the node in the pattern with the fewest legal mappings in the data and creating a partial match for those mappings. It expands a partial match by selecting an unexplored node mapping $(PatternNode, DataNode)$ and generating new mappings for each link adjacent to $PatternNode$ to every mappable link adjacent to $DataNode$. When a pair of links is mapped, the nodes on the other ends of those links are mapped as well.

The search selects as the next state to expand the one with the minimum worst-case cost—that is, the cost of the mappings so far plus the cost of deleting all unexplored nodes and links in the pattern. Since the cost of deleting all unexplored nodes is guaranteed to be an upper bound on the eventual cost of any extension to that partial mapping, this selection heuristic meets A*'s admissibility criterion, and the search is guaranteed to find the lowest-cost solution. The search prunes any partial match that cannot possibly have a lower cost than the best n matches found so far, where n is the maximum number of matches the user wants to see, as well as any that cannot possibly have a lower cost than the pattern's maximum allowable cost. In addition, at the end of the process LAW prunes any mappings that are *subsumed* by other discovered mappings. A mapping A subsumes a mapping B if $MappedEntities(A) \subseteq MappedEntities(B)$, that is, if they differ only in that B has more node and link deletions than A.

The search process is designed to find a good set of pattern matches quickly, and then use those existing matches to prune the remainder of the search. One key asset of the approach is that it is an *anytime* algorithm: at any point during the process the algorithm can return the set of matches it has found already, and that set of matches will monotonically improve as the process continues.

Figure 4 shows the benefit of different aspects of the approach. The top line shows the amount of effort it takes to match data sets of varying size with relatively little search control. The middle line shows the performance of branch-and-bound—that is, pruning the search as described above, but selecting the next state to explore in depth-first order rather than a heuristic evaluation function. The bottom line adds the evaluation function selection to convert the branch-and-bound approach to A*. One thing the graph demonstrates is that the match time is not completely determined by data set size; both the heuristic approaches were slower in the mid-size data set than they were in the largest one. The match time will be dependent on a large number of factors in addition to data set size, including the size of the pattern, the number of good partial matches in the data, and the pattern-defined maximum allowable cost of a match.

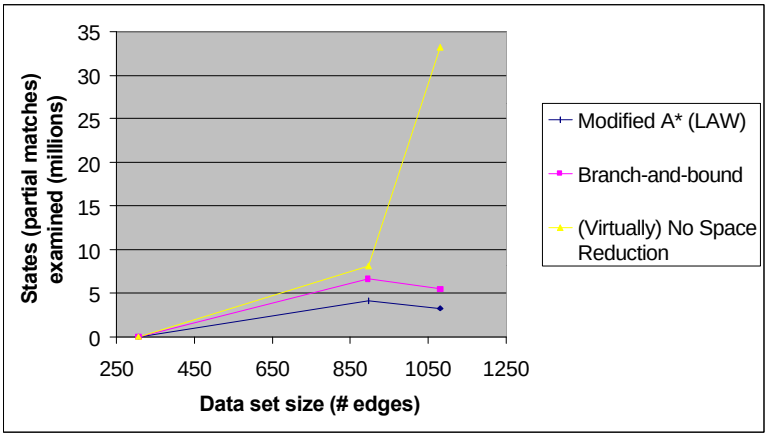


Figure 4: Effects of heuristics in pattern search

2.4 Experimental Results

One of the key challenges for a useful intelligence tool is to scale to problems of realistic size. For LAW, that means finding partial matches to graphical patterns representing realistic scenarios in large collections of relational data. Here, "large" may mean ten million relations or more. Since subgraph isomorphism—a simpler version of the problem that LAW's matcher is solving—is known to be NP-complete, it is important to evaluate LAW's ability to process patterns and (especially) data sets of realistic size.

To evaluate LAW's scalability, we ran a set of experiments measuring the effect of data set size on LAW's match time, and the effect of various advanced features of LAW—specifically, hierarchy and cardinality—on LAW's match time. The results of these experiments showed encouraging progress toward the goal of solving realistic problems in a reasonable amount of time, and at the same time they indicate areas of needed improvement in future work.

The experiments were designed to measure how LAW's match time changed as various problem characteristics were varied. We used IET's data set generator to create data sets of various sizes, where size is measured by number of relations in the data. We kept the other settings of the data set generator—observability, noise, and so on—constant and on the "easy" settings. We ran the set of experiments on a small suite of patterns of various characteristics. Individual experiment descriptions below will describe the specific patterns used. The dependent vari-

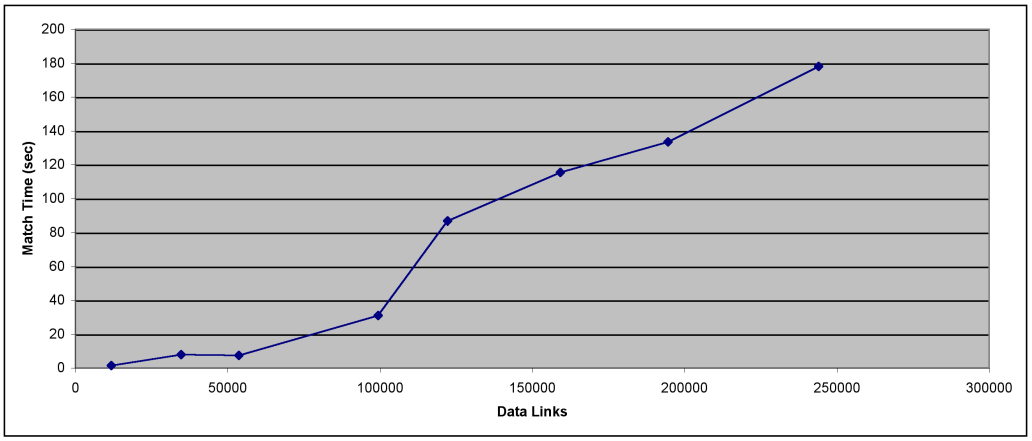


Figure 5: Growth of match time with data set size

ables we measured were (1) CPU time elapsed during the matching process, and (2) the number of A* search space states examined during the matching process. Our goal was to measure (in separate runs) both the overall impact of data set size on LAW's match time, and to measure the contribution of individual features of LAW and its pattern language toward LAW's scalability.

2.4.1 Scalability

The first experiment was designed to measure LAW's ability to scale to data sets of realistic size. We used LAW to match a pattern representing two cooperating threat groups—two groups that share a member in common, each carrying out threat activities—against data sets of sizes ranging from approximately 12,000 links to more than 240,000 links. This pattern is not the largest in our pattern library, but it is one that we feel represents realistic computational challenges for LAW's algorithm.

The match time results are shown in Figure 5. LAW's match time ranged from just over a second for the smallest data set tested to just under three minutes for the largest data set. While the graph is not a perfectly straight line, it does not give us any reason to think that LAW's match time is super-linear in the size of the data set.

If we assume that the match time is linear in the size of the data set, we can extrapolate from these results the time it would take LAW to match this pattern

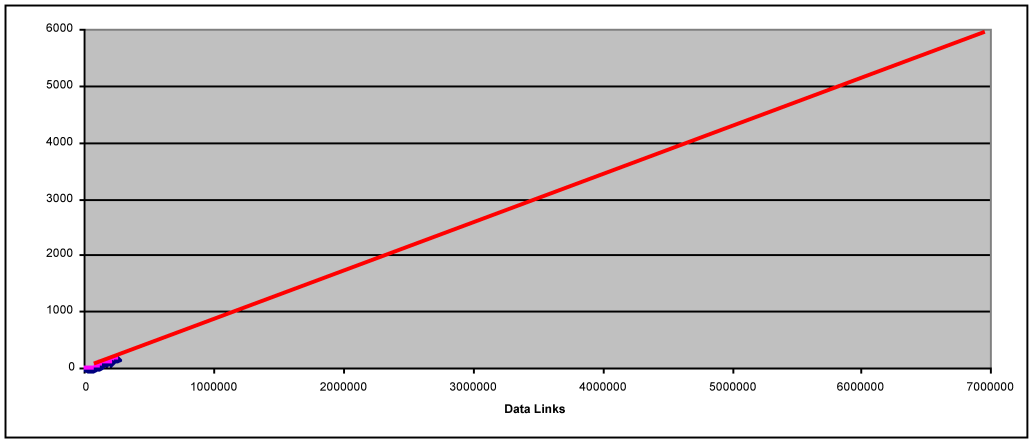


Figure 6: Growth of match time with data set size, extrapolated to very large data sets

against a very large data set that could be encountered in actual intelligence work. Figure 6 shows the rate of growth shown in Figure 5 extended to a data set with 7,000,000 links. The number 7,000,000 was chosen based on the size of the largest data set used in the TIA experiments for which statistics were made public—the "AVRS with GDB Extensions data set."[5]

If the linearity assumption holds, Figure 6 shows that LAW would take about an hour and forty minutes to match this pattern in a very large, realistic data set. Despite the fact that we envision that LAW will often be run in modes where immediate response is not required (e.g., matching a set of patterns in an overnight run), our belief is that this is too long for an information gathering tool for real-world intelligence analysis. Our experience leads us to estimate that widespread user acceptance would be possible only if the matching time on large data sets is around one-tenth the estimated time shown in that graph—that is, around ten minutes rather than one hundred. Thus, this experiment identifies a critical immediate goal for our ongoing work on LAW: a further order-of-magnitude speedup on matching realistic patterns in large data sets.

[5]Statistics for this data set indicated that it had only 152,000 links, but also 7.1 million attributes. We base our extrapolation on the assumption that attributes would be converted to links for matching in LAW.

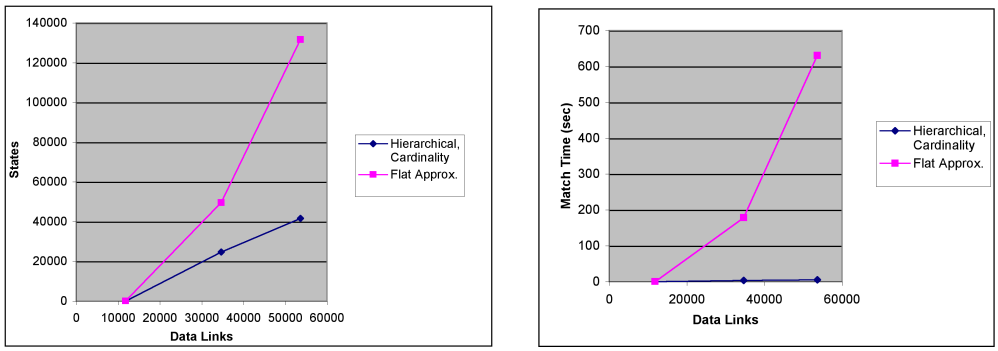
2.4.2 Cardinality Results

The second experiment was designed to measure the extra efficiency achieved by adding cardinality to the pattern language and matching capability. The cardinality constuct in the pattern language allows the pattern designer to impose numerical constraints on the number of matches of a subgraph—for example, "three or more meetings." More important for efficiency, it also provides a way of grouping results of a subgraph match and thereby controlling the combinatorial explosion of possible matches. That is, instead of creating a separate new parent graph match for each subgraph match, the cardinality construct allows LAW to group all matches to a subgraph under a single parent graph match.

In these experiments, we compared the resources used—measured by search space size and time consumed—by the LAW matcher in matching two patterns: (1) a hierarchical graph with cardinality constraints on its subgraphs, and (2) the best "flat" approximation of graph (1). The best flat approximation was determined by inserting N copies of each subgraph into the parent graph, where N is the value of the cardinality constraint on that subgraph.

It is important to note that the flat approximation (2) is truly an approximation to the hierarchical graph (1), and is *not* equivalent, either semantically or computationally. In other words, the fact that we achieved efficiency gains by using hierarchical graphs is not unexpected, and does not represent a breakthrough in complexity theory. Rather, these experiments measure the scalability benefit of giving the user the extra expressive power to represent what he wants. Situation descriptions that include "N or more" occurrences of an event are common, both in the EELD Challenge Problem domains and in the real world. By giving the user the ability to represent such situations, and by giving the matcher the capability of matching those situations efficiently, we expect to see improved efficiency compared to matching inexact approximations.

Figure 7 shows the efficiency gain from hierarchy and cardinality on one pattern, representing a threat group with two or more members each acquiring a threatening resource. The hierarchical pattern contained a single subgraph with a "2 or more" cardinality constraint. The graphs show the change in search efficiency as the data set size grows, with the smallest data set size containing around 15,000 links and the largest containing around 75,000. Figure 7a) measures the number of search states explored, which was reduced by a factor of more than three using hierarchical graphs on the largest data set. Even more encouraging is the amount of time saved by hierarchical patterns, shown in Figure 7b. Hierarchical patterns with cardinality improved match time by over two orders of



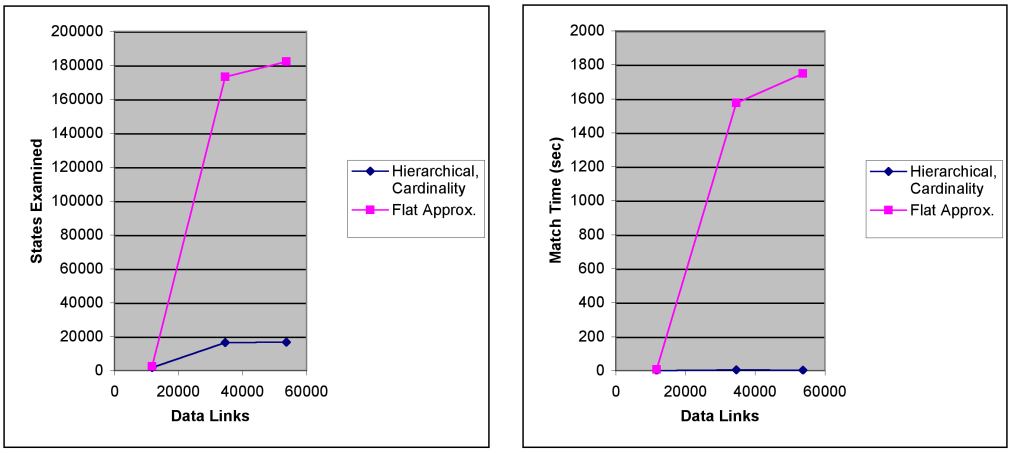
(a) (b)

Figure 7: Improved efficiency from hierarchy and cardinality on "Group Gets Resources for Mode" pattern, measured by (a) amount of search space explored and (b) match time

magnitude—from 630 seconds to six.

Figure 8 shows the same measurements for another pattern (and its flat approximation). This pattern represents hub-and-spoke communication—two or more phone calls from members of a threat group to another "hub" group member. Like the "Group Gets Resources for Mode" pattern of Figure 7, the hierarchical version of the hub-and-spoke pattern contains a single subgraph with a "2 or more" cardinality constraint. The efficiency gain with this pattern was even more dramatic than that shown in Figure 7. Figure 8(a) shows a search space reduction on this pattern of over an order of magnitude—from 180,000 states to 16,000—and Figure 8(b) shows a match time reduction of almost three orders of magnitude—from over 1,700 seconds to less than three.

The dramatic difference between the search space reduction and the search time reduction—the (a) and (b) graphs in each figure—is an interesting phenomenon, and one for which we do not have a definitive explanation. Our current hypothesis is as follows. State expansion in LAW's A* search involves copying the parent state. When the patterns are hierarchical, the pattern representation is more compact—since each subgraph is represented in a parent graph as only a pointer, rather than a complete list of all nodes and links in the subgraph—and therefore the copying operations per state expanded are smaller.



(a) (b)

Figure 8: Improved efficiency from hierarchy and cardinality on "Hub-and-Spoke" pattern, measured by (a) the amount of search space explored and (b) match time

2.4.3 Caching Results

The final set of experiments measures the value of caching subgraph matches in hierarchical pattern matching. We designed a mechanism in the LAW matcher that caches subgraph match results during a match of a parent graph. Subgraph matches are cached indexed by

- The subgraph being matched, and
- The mappings of the subgraph's interface nodes (see Section 2.1).

Once a subgraph S with a given set of interface node mappings $< m_1, \ldots, m_n >$ is matched, the results for any future attempts to match S with $< m_1, \ldots, m_n >$ will be retrieved and returned from the cache.

Our hypothesis was that caching would be especially useful for patterns that contain multiple copies of the same subpattern. Such a pattern is shown in Figure 9. This pattern represents two different Threat Groups, related through a common member, each acquiring a threatening resource. Matching this graph will require multiple attempts to match the AcquiringThreateningResource subgraph for each Person, and caching will eliminate all but one of those attempts.

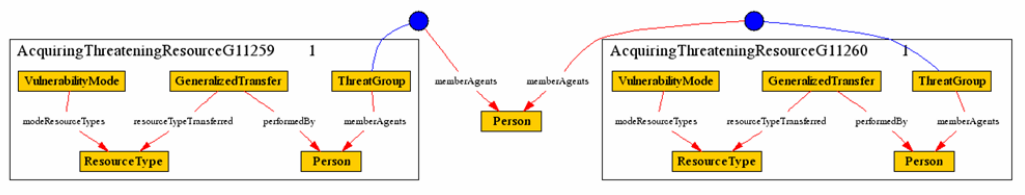


Figure 9: "Two Related Groups Acquiring Threat Resources" pattern

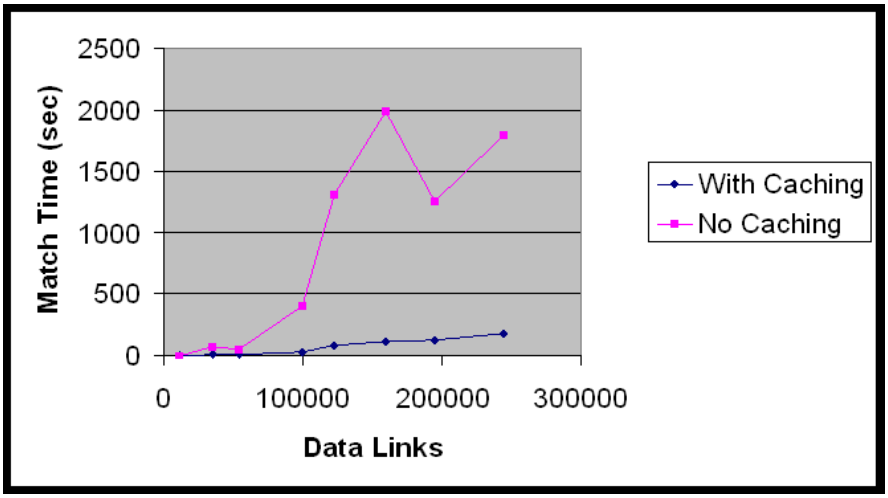


Figure 10: Effect of caching as data set size grows on "Two Related Groups Acquiring Threat Resources" pattern

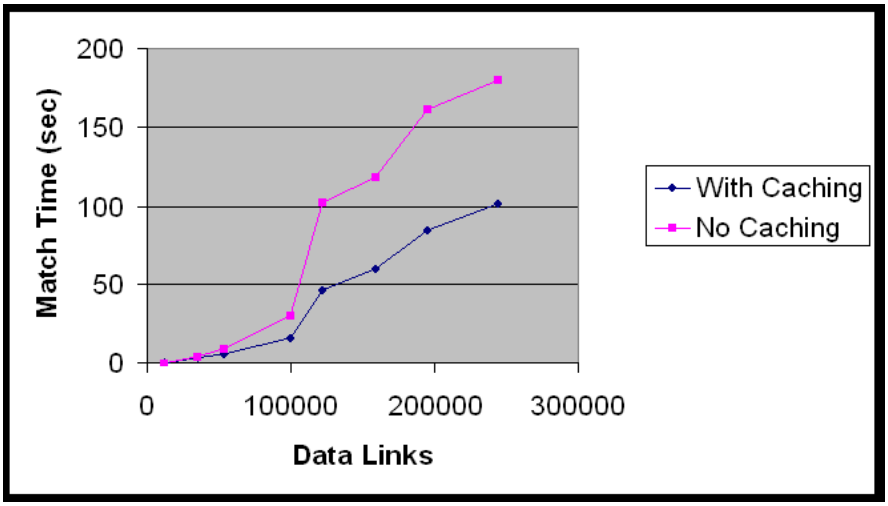


Figure 11: Effect of caching as data set size grows on “Group Gets Resources for Mode” pattern

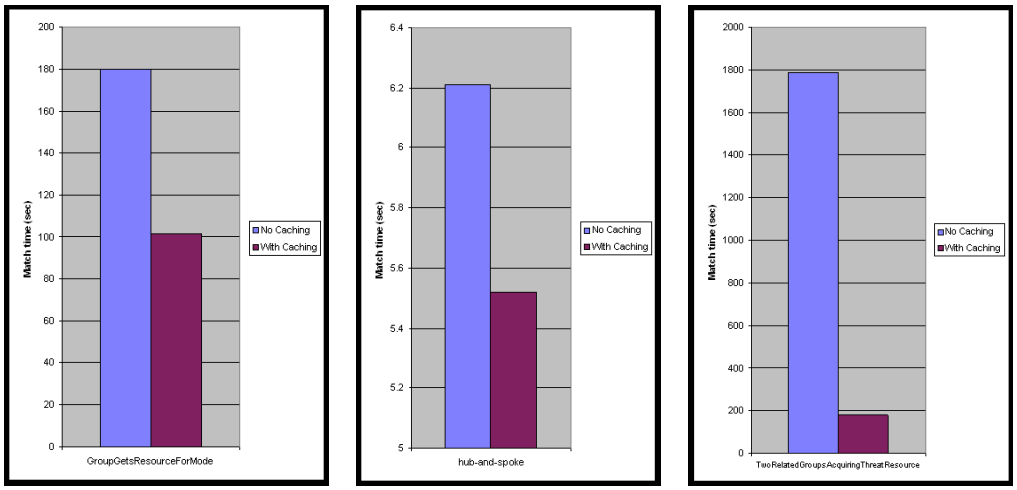
As we expected, Figure 10 reduces match time dramatically for Figure 9’s pattern. For the largest data set tested, caching reduced match time by an order of magnitude, from 1,800 seconds to 180, and the time savings were even greater for some smaller data sets.

However, we did not expect the results shown in Figure 11. The pattern that generated those results—“Group Gets Resources For Mode”—is a pattern with a single subpattern, and one for which we expected little or no repetition. Our hypothesis was that caching would have little impact; in fact, we thought it would be possible that the overhead associated with caching could cause the runs with caching to be slower than the ones without. Instead, Figure 11 shows that caching did have a significant positive effect even for this pattern, cutting match time roughly in half for all data set sizes tested.

Figure 12 summarizes the caching results, showing the benefit of caching for all three patterns tested on the largest data set tested.

2.5 Metrics for Approximate Pattern Matching

Much of the pattern-matching work within the LAW implementation is grounded in a theoretical model of approximate pattern matching. This model is described in detail in the Appendix, and we provide an overview of it here.



(a) (b) (c)

Figure 12: Summary of caching results: benefit of caching on largest data set for (a) “Group Gets Resources for Mode”, (b) “Hub-and-spoke Communication”, and (c) “Two Related Groups Acquiring Threat Resources” patterns

The complex patterns considered by the matching techniques discussed in this report may be regarded, from a logical perspective, as axioms that specify a *theory* while the objects being sought are *models* of that theory. Informally, the database may be thought of as a collection of objects that are linked by various predefined relations. At a more formal level, facts describing the existence of these objects and their relationships are expressed as the conjunction of the members of a set of instantiated logical predicates. Patterns may also be conceived in terms of logic constructs, requiring the existence within the database of certain instances of objects and that of links, or relationships, between them.

In the theory's treatment of patterns, we generalize the notion of a pattern by regarding them as specification of *elastic constraints* on potential models. This type of specification, which is familiar in information-retrieval applications, permits ranking of instances of data structures by their degree of matching with the ideal conditions. For example, a requirement to match the pattern "*Person* P is a *Southern European* who *breeds attack dogs*" might be matched, albeit not perfectly, by an object of the type `Person` who was born in *Central France* (which is close to and overlaps *Southern Europe*) and who *keeps* (but it is unclear whether or not he *breeds*) *wolves*.

In this extended view, patterns do not express strict requirements that are either met or not met. Rather, patterns should be regarded as procedures that rank the adequacy of alternative variable-to-object assignments as potential solutions of a database-retrieval problem. Correspondingly, the values of properties of objects in databases (e.g., *Southern European*) and the nature of the properties themselves (e.g., *breeds*) should be regarded as elastic descriptions that may be met to various degrees. Each possible instantiation matches the pattern to some degree, expressed by a number between 0 and 1 that measures the extent to which such an instance matches the pattern specification. Pattern instances that strictly match, in the logical sense, the pattern specifications have a degree of matching equal to 1, while semantically unrelated instantiations—"unrelated" in a sense that we will formalize in the Appendix—have a degree of matching equal to zero.

Patterns may be regarded, therefore, as mechanisms to measure the *distance*, *similarity*, or *resemblance* of potential solutions of a matching problem to a prototypical set of *ideal* or *perfect* matches. This similarity function, which reflects the semantics of the specific problem being considered, is the basis for the definition of numerical measures of *degree of matching*. This conceptualization of the pattern-matching problem suggests that it may be treated as a generalized logical program, that is, as a procedure to search a space of potential solutions and to rank their suitability [36, 18], which extends the theorem-proving approaches of

logical programming [13].

The theory defines a degree-of-matching function in terms of the *degree of admissibility*, or *adequacy*, of the modifications required to transform a database into a modified counterpart that strictly matches the pattern. Such a best match may be informally described as having the largest admissibility value (i.e., lower transformation cost) among all transformations leading to transformed databases that match the pattern from a classical-logic viewpoint. Database transformations are defined as the composition of a sequence of certain *basic edit operations*. Each edit operation is associated with a numerical value gauging its admissibility. The admissibility of a particular transformation is then defined as a function of the admissibility of its component edits.

The theory's metrics determine the extent to which a database matches the specifications of a pattern, and are based on semantics provided by knowledge structures such as ontologies. These metrics are intended to be employed in connection with graph-based approaches [41, 8, 48] to database representation. Graph-editing techniques provide a useful framework to describe a variety of complex objects while permitting their comparison in terms of the extent (or cost) of the modifications that are required to transform a graph-based representation of one object into another. These techniques have considerable generality and may be applied to a wide variety of problems. In each application, however, it is necessary that the functions employed to estimate the admissibility of graph transformations reflect the particular meaning attached to each editing operation. The model detailed in the Appendix is devoted to the derivation, from the perspective provided by a combination of logical and metric perspectives, of specific admissibility measures, called *database-editing metrics*, applicable to pattern matching in databases.

3 Example and Interface

This section has two purposes. First, it provides an example of the user's interaction with LAW—and especially the LAW pattern matcher described in the previous section—through screen shots of the system. Second, it discusses the design of LAW's user interface, which is based on the common pattern framework and Web services architecture described in Section 4.

3.1 Interface Design

LAW's interface runs primarily through the Web. The Web-based implementation offers two main advantages. First, any connected computer with a browser can access it without the need for an error prone installation cycle. Second, it cuts to zero the time from release to deployment, and it allows concerns of the users to be answered, implemented, and delivered much more quickly. On the other hand, the disadvantage of a browser-based interface is that there is only so much that can be done using HTML and JavaScript. Highly interactive HTML interfaces are either kludgy and inefficient or impossibly slow. For that reason, certain highly interactive pieces of the LAW interface—the pattern editor in particular—are implemented as local applications that connect to the original server using Web services.

Given the pattern-matching task described above, a user interface must provide a way to make sense of the structure and amount of the information available. It must also provide a pattern editor that permits the creation of patterns in terms of the data. The most important component, the pattern matcher has limited user interface. When matches are extracted, they must be shown in a drillable way that is consistent both with the views of the data and the pattern construction.

3.1.1 Visualization

Visualization in LAW can be divided into three areas: patterns, the data to which the patterns are matched, and the ontological concepts of which the patterns and the data are composed.

As described earlier, patterns contain two elements: a graph that represents the prototype situation, and the edit distance parameters that represent the allowable deviations from the prototype. LAW's presentation of patterns displays both elements. Figure 13 shows LAW's display of a pattern. The display shows the objects and relations of the pattern (the nodes and links) as well as the edit distance parameters (encoded in colors of the nodes and links).

Ontology management is a well-studied problem [30]. Since ontologies are not a primary focus of LAW, its ontology exploration and editing capabilities are limited. Figure 14 shows the user exploring the ontology via LAW's hierarchical browser. Although the range of things that can be done to the ontology is restricted, our tool can import DAML [23] or CycL [15] ontologies, so modification of these is permitted by any tool that is able to export in those formats.

The last area, visualization of primary data, did not play an important role in

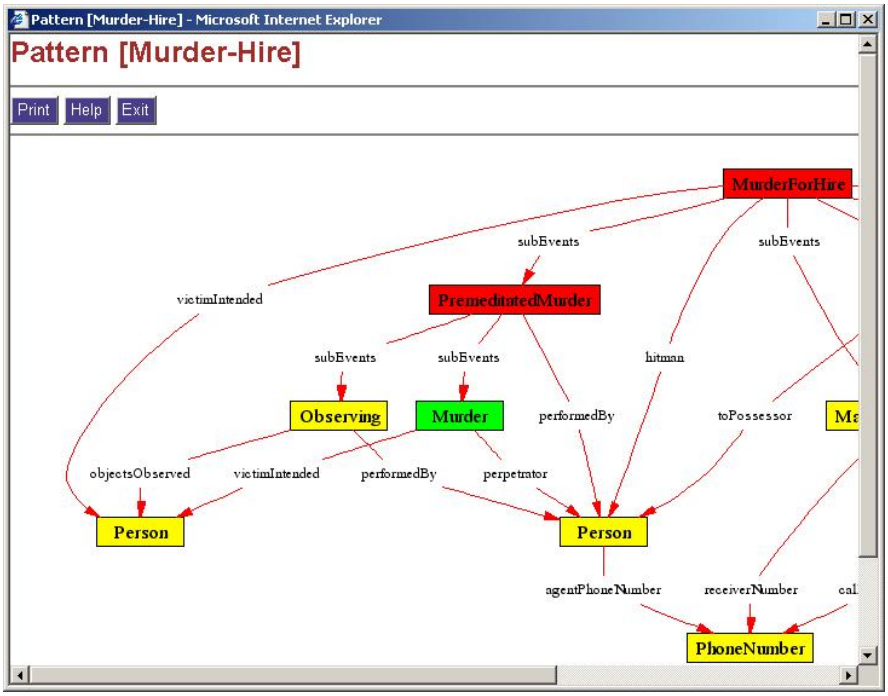


Figure 13: LAW's display of the Murder-for-Hire pattern

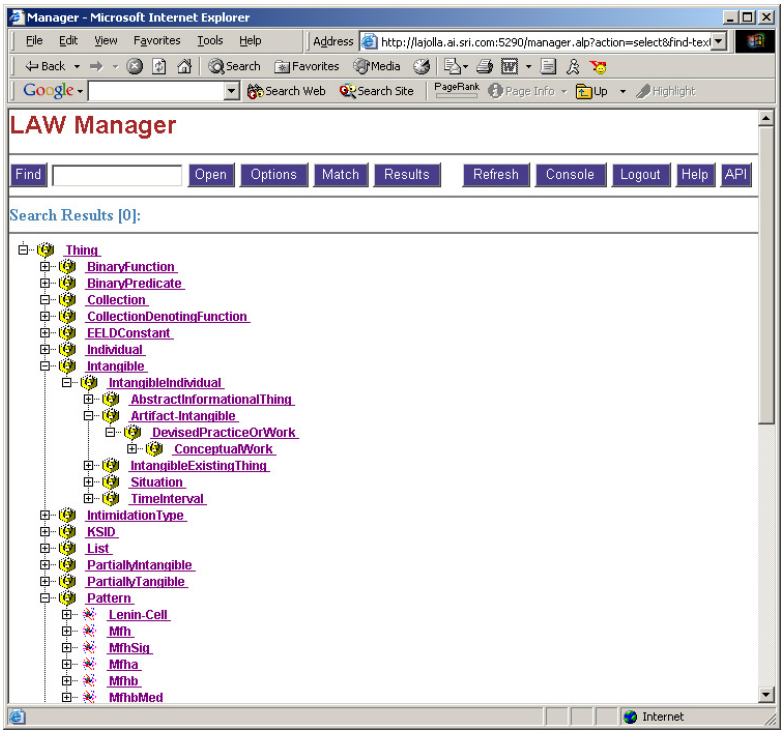


Figure 14: Ontology browsing

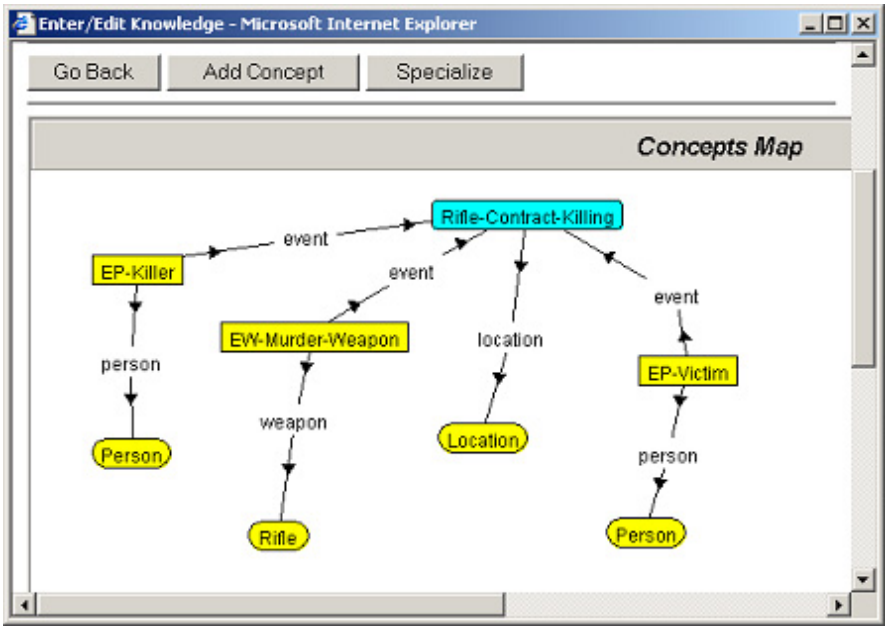


Figure 15: SHAKEN interface

our work on this project.

3.1.2 Pattern Editing

LAW's development has been guided by our view that pattern matching should be an interactive task, where patterns will be continuously refined during an iterative matching process. Some patterns will even have to be created either from scratch (i.e., directly from concepts in the ontology) or from other patterns.

The interactive nature of the pattern-matching process requires that the user be able to revise his thinking quickly, which in turn requires the ability to make fast modifications to the graph. A browser is ill-suited for this type of interaction. In the current version of LAW, we are using a knowledge acquisition tool called Shaken [46] for the construction of patterns. Figure 15 shows the user editing a pattern in Shaken.

The requirements for a pattern editor include (1) the ability to edit and/or incorporate existing patterns coming from other sources, (2) the ability to build patterns from concepts in the ontology, and (3) the ability to handle a wide range of pattern sizes, ranging up to thousands of nodes. The pattern editor is basi-

cally a graph editor, but with a constrained set of possible operations. It is also component-based, which means that patterns can be made of other patterns. The main operations that can be performed on a pattern are the addition of a node, the connection of two nodes, and the merging of two nodes.

3.1.3 Pattern Match Visualization

The final piece of the interface is the pattern match (result) visualization. Just as we emphasized how important it is that we can observe the data and the patterns under the same framework, the same is true for the results. This is important so that the relationship between the pattern and the results instantiating it is self evident. If a mental mapping between one and the other is difficult, then the ability to modify the pattern as we look at the results is diminished.

When a request for matches is initiated, the server will asynchronously start the search and inform the user of it. When the results are available, the ones that meet the user's criteria are presented, ordered best to worst. The images of the results are small replicas of the graph used to describe the pattern. The nodes of the result image are colored to display the degree of accuracy obtained in the match. Figure 16 shows the results of the match of Figure 13's pattern.

Although not implemented currently, the idea is that these matches can be reentered into the system as data, so that they can be revisited later. This scenario makes sense in the case of streaming data, where the system does not yet know if more evidence is going to become available. There must be a big portion of the system devoted to what we call "hypothesis management"—that is, storing, revising and comparing past results. The current system offers a good explanation of why a match coincides with the pattern offered, by assigning scores to all the nodes and relationships.

4 Architecture

While the LAW pattern matcher described in the previous sections provides a powerful and flexible mechanism for finding partial matches to patterns, it cannot possibly cover all the pattern-matching criteria an analyst could have. There are many tools under development, general and specialized, that support the many complementary data exploration requirements of the intelligence community. In a given search session, an analyst may want to combine results from an approximate match to a general scenario (as retrieved, e.g., by the LAW matcher just

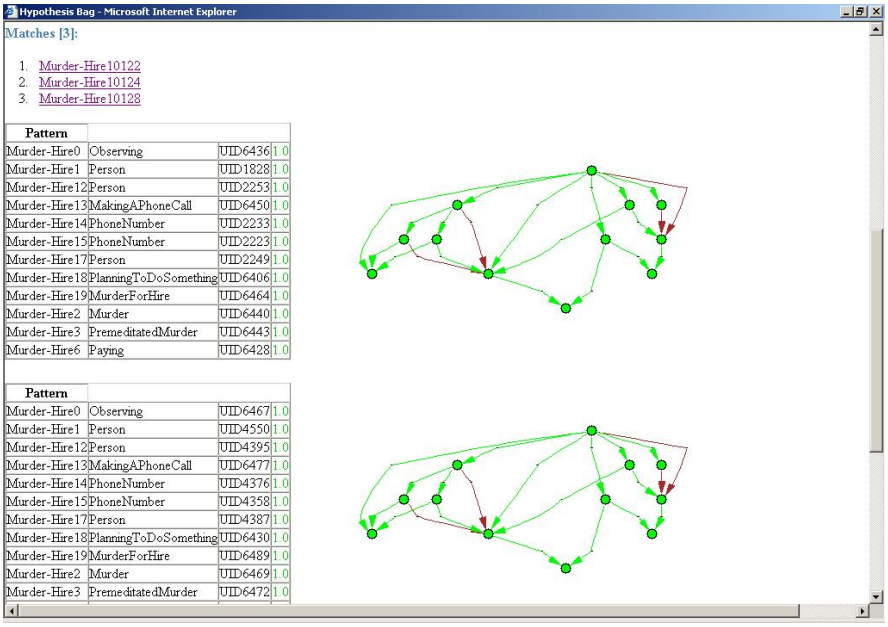


Figure 16: Results list page

described), a general probabilistic reasoning engine (e.g., [44]), and a specialized tool for determining group membership (e.g., [24]).

We have developed and implemented an architecture for LAW that supports integration of multiple pattern-matching tools. It includes a language for sharing patterns, and a Web services architecture that makes integration straightforward. The LAW user interface allows a user to interact with these tools through a common framework. Right now, the user must select and task the tools manually, but our aim for the future is to have the system provide support for tasking of tools through LAW's Control component.

4.1 High-level Architecture

LAW is architected as a Web application, made up of a number of Web service components. At the heart of LAW is a knowledge base server, with an in-built Web server. Data for this knowledge base server can reside in either a database or a flat file. The client side of LAW is ephemerally generated Web pages, encoded in HTML, Javascript and embedded Java applets, with the client being any standard

modern Web browser.

This thin client, Web server architecture had previously been used for a successfully fielded collaborative structured argumentation application, SEAS [25]. The adoption of a Web browser as the client, removed the need to install and maintain client software in end-user organizations. For LAW, we decided to enhance this architecture, using a Web service model. Our driver for this was that partner companies were simultaneously developing applications, which we wanted to make available to the LAW user through its GUI. Likewise, several of the LAW components were of general value to external partners, outside of the LAW framework. Given that we were not able to state at the outset all the different ways in which we and other partners would integrate the various components into larger systems, Web services offered an ideal way of providing a lightweight, flexible means of integration. The idea was that during the development of the various components, and as our understanding of the domain was refined, we would be able to experiment with different configurations of Web services to provide different, complementary, end-user solutions.

From a user's perspective, the configuration/location of the underlying services, such as pattern matching, are not visible. Instead the user is presented with an interface, which can hide all the integration details.

Figure 17 depicts the current LAW architecture. In the current architecture, LAW can be viewed as having three internal components:

- the control component that is responsible for coordinating the tasks between the user input (via UI) and the other (internal and external) components.
- the UI component, that is the user interfaces that are presented to a user to request pattern matches and to view the results of matches
- the Pattern Match component

LAW itself contains a Web server, which is used to communicate between the different internal components, as well as external components, with the messages being encoded in XML, conforming to defined schema (PatternML, HypothesisML and ControlML). External components are all intended to be tasked using SOAP, although currently some components are still tasked through custom interfaces.

The LAW server sits on top of SRI's knowledge base server, with communication between LAW and the server via OKBC. The knowledge base server contains the domain ontologies used by the evidence data sets that are to be matched against.

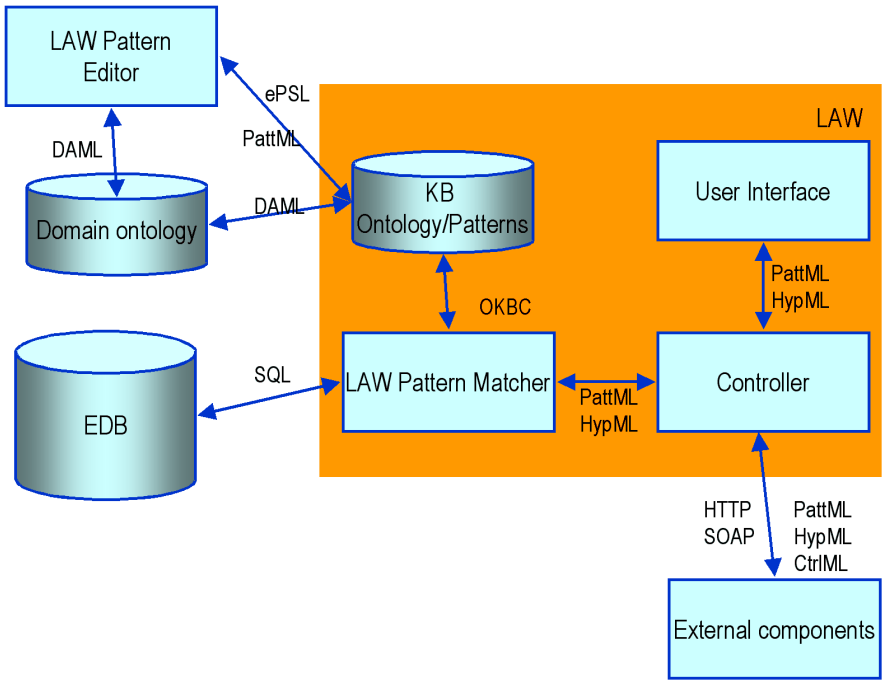


Figure 17: LAW architecture

Initially, LAW was implemented with the evidence data sets to be matched also contained within LAW's knowledge base server. This was done mainly for efficiency, as a placeholder while the EELD program was developing a database schema for evidence (the EDB schema). Since the EDB schema was developed, LAW now interfaces directly via SQL with EELD evidence databases, stored in a separate mySQL relational database.

The LAW pattern editor is still a separate application from LAW. Originally developed using SRI's Shaken editor [46], LAW's current pattern editor was rewritten as a pure Java application. The pattern editor uses the DAML form of the EELD ontology as input, and produces patterns as output in two formats—PatternML and ePSL (the program-wide pattern specification language developed in 2003).

4.2 SOAP

We chose to adopt SOAP (Simple Object Access Protocol—www.w3.org/TR/SOAP) as the standard communication protocol between LAW's internal and external Web services. SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol, which is neutral about the actual transport mechanism. This means that HTTP, SMTP, raw TCP, an instant messaging protocol like Jabber, and so on, could potentially be used for message transport. For LAW we decided to adopt HTTP as the communication protocol. LAW uses SOAP to enable remote procedure calling (RPC), where an RPC call maps naturally to an HTTP request and an RPC response maps to an HTTP response. SOAP was also attractive from an architectural standpoint in that it does not require that a message be sent from a client to a server in a single "hop." The SOAP specification defines the notion of intermediaries—nodes that a message passes through on its way to its final destination. Intermediaries can be used to "virtualize" physical network topology so that messages can be sent to Web services using whatever path and whatever combination of transport protocols is most appropriate. This last capability facilitates our vision for the LAW architecture as a federated collection of services that communicate via networks to coordinate their search for pattern matches over distributed data sources, in service of a community of intelligence analysts.

5 Integration with Other Pattern-matching Components: TIEs

A series of technology integration experiments (TIEs) was conducted over the course of the EELD program, with the aim of evaluating how an integrated EELD system performs on a challenge problem data set. SRI's LAW tool took part as a component in several of these experiments. Details of these experiments are given below. The common element in all the experiments was the adoption of a lightweight integration model, using Web services and XML schemas to communicate between the different components. The schemas used in these experiments are first described, followed by the details of the experiments.

5.1 XML Schema

To allow communication between various components internal to LAW and external component Web services, there was a need to design XML schemas for the content of the SOAP messages, so that all components could understand the syntax of the content, and map the content to their internal representations.

We adopted a layered approach to schema design, where the core is PatternML, a schema to describe template domain patterns [21]. PatternML was designed to have two main uses: as an interchange language between pattern editors/visualizers, and as an input format to pattern-matching components. Patterns defined using PatternML make reference to existing ontologies for domain concepts.

Layered on top of PatternML is the HypothesisML schema, which was designed to be a common output format for pattern match components and as an input format for hypothesis visualizer or hypothesis management components [20]. HypothesisML describes the match between a PatternML pattern and a particular evidence data set.

Layered on top of both of these schemas was the ControlML schema [19], which was designed to describe intended task control information. ControlML allows one component to describe what it wants other components to do, together with various control parameters, and allows PatternML or HypothesisML as content (to describe pattern templates or hypothesized pattern match(es), respectively).

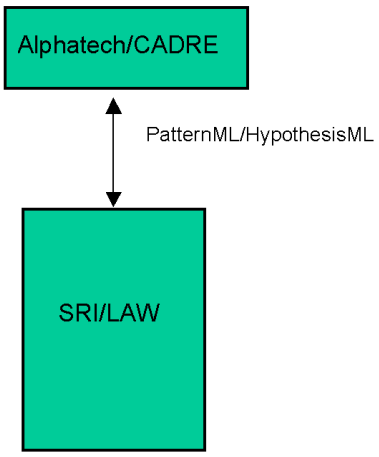


Figure 18: LAW's role in TIE1

5.2 TIE Implementations

5.2.1 2002: TIE1

The LAW server provided visualization services for the TIE1 experiment led by Alphatech. The visualizations were of patterns and hypotheses. Alphatech's CADRE system interoperated with SRI's LAW server via defined APIs. Figure 18 shows the experiment architecture. The goal of the experiment was to demonstrate how a Web service architecture could support the rapid integration of components from different contractors, to provide services currently unavailable in a tool (in this case visualization services). By defining and agreeing on XML schema for patterns and hypotheses, and using standards-based communication protocols (http in this case), a service was quickly developed. ControlML and SOAP services were not developed in time for the experiment, so we used http POST to communicate between CADRE and LAW.

The experiment consisted of two parts:

1. CADRE sent an http message to LAW to display a PatternML file (pattern file sent along with http message). LAW returned an HTML page displaying a graphical representation of the pattern.

2. CADRE sent an http message to LAW to display a HypothesisML file (hypothesis file sent along with http message). LAW returned an HTML page displaying a graphical representation of the hypothesis.

The experiment required CADRE to provide a translator to translate from its own internal pattern and hypothesis representations to the common pattern representations PatternML and HypothesisML. CADRE also required the ability to send requests, using agreed control messages for the experiment. LAW, too, required a translator from PatternML and HypothesisML into LAW's own internal pattern and hypothesis representations. LAW already possessed the API to produce HTML pages that visually displayed the content of the PatternML and HypothesisML files. The experiment was a success, in that Alphatech was able to demonstrate accessing the LAW server to provide visualization services.

5.2.2 2002: TIE3

The domain for the 2002 challenge problem was Russian Contract Killing (RCK). A series of artificially generated data sets were provided by IET, which contained fictitious data from different data sources, such as newspapers, police reports, direct observations, telephone records, and bank records. The goal of the evaluation was to identify those murders that were contract killings, and to identify the people involved and the details of the murder (communication events, money transferred, location, and business involved). The output from the TIE was scored against ground truth to evaluate performance.

The architecture of TIE3 is given in Figure 19. LAW acted as the hub of the TIE, providing user interaction, control (component tasking) pattern matching and hypothesis management services. Other contractor components provided services such as pattern matching (Alphatech, CMU, CHI) and accessing external data sources (USC/Fetch).

Like in TIE1, XML schemas were used as the interfaces between the different components. Again, due to time constraints, all participants did not use SOAP and ControlML. CMU's group detection algorithm (GDA) was called directly from LAW's pattern matcher, using GDA's C-based API. USC/Fetch's city location service was called from LAW's pattern matcher using its HTTP POST interface. Alphatech and CHI's pattern matchers were called using SOAP and ControlML messages. To ease integration, the PatternML version of the RCK pattern was not actually sent from LAW to the pattern matchers. Instead, an agreed pattern name was passed, and each individual pattern matcher component used its own internal

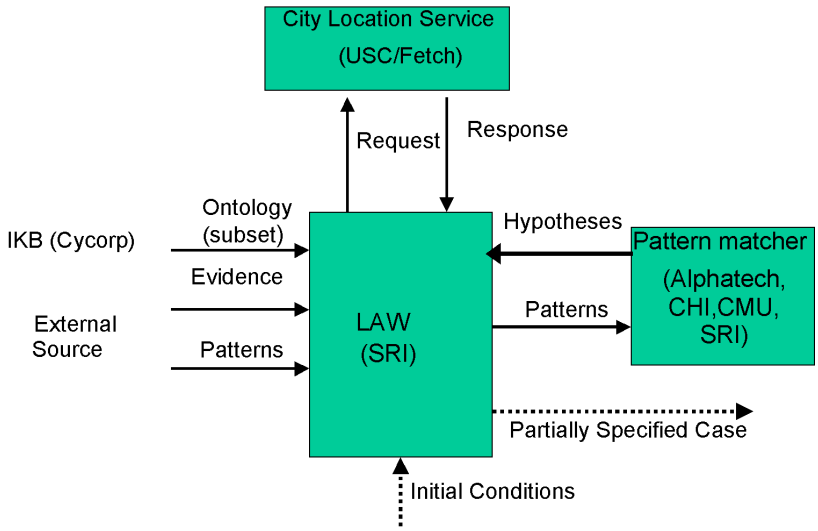


Figure 19: TIE3 architecture

representation of the RCK pattern for the matching. However, both Alphatech and CHI did return HypothesisML hypotheses back to SRI, which required them to develop translators from their internal hypothesis representations to HypothesisML. SRI's LAW component provided the client interface, allowing a user to choose a data set to match against and the particular pattern match component the user wanted to make the match (including LAW's own pattern matcher).

Hypothesis merging was not attempted in TIE3. Instead, TIE3 returned multiple results for each data set, for those cases where the different pattern matchers were able to find some matches. Before returning results for TIE3 to the evaluation team for scoring, LAW translated the results from HypothesisML to the evaluation team's own XML-based result schema.

The integration experiment was a success, in that it demonstrated how using a standards-based XML schema and communication protocols allowed diverse components to integrate in a short time.

5.2.3 2003: oddTIE

The domain for the 2003 challenge problem was changed to be an abstract threat scenario domain, where members of threat groups apply capabilities and acquire

resources necessary to carry out threat events. As for the 2002 challenge problem, a series of artificially generated data sets was provided by IET, which contained fictitious data from different data sources, primarily telephone and purchase transaction data. The data sets varied along several different domain dimensions, like size, data observability, corruption of data, and connectivity. The goal of the evaluation was to identify the threat groups in the data set, their members, and the threat events that they carried out. The output from the TIE was scored against ground truth to evaluate performance.

SRI's LAW system was part of the oddTIE, which was an outgrowth of TIE3, expanded to contain pattern learning (PL) technologies, since the 2003 evaluation was to be of combined link discovery (LD) and PL technologies. OddTIE was characterized through its use of specialized LD components for group and event detection/refinement, and by its use of multiple threads to exploit the strengths of different components. The results were then merged using hypothesis management algorithms to perform smart merging, hence improving what any single approach could have achieved by itself. As in TIE3, Web-based integration of components was used, and SRI's LAW provided the component tasking/process control that made the integration threads work. PL technologies were used in three different ways: 1) to perform traditional offline pattern learning, which was then utilized (either directly or indirectly) by LD components; 2) to provide specialized online matching; and 3) to provide online hypothesis ratings.

The overall architecture for oddTIE is given in Figure 20. This diagram does not show the finer detail of the pattern matching or hypothesis management tasks. ISI/USC, UMass, NYU and Stanford all contributed to the group detection/refinement part of pattern matching. Alphatech, SRI, and CHI all contributed to the event detection/refinement part of pattern matching. NRL provided the rating and selection functions of hypothesis management.

The oddTIE results were mixed. Overall in the EELD program, oddTIE produced the best group detection (as measured by evaluation score) of any TIE, with precision and recall scores of 99% and 74%, respectively. However, the results were to us modest, but we were pleased that the technologies did do best on the harder data sets. In general we believe we were too parsimonious in our use of secondary data, and data observability affected group detection much more than perhaps was intended by the evaluation team. OddTIE's event detection was not particularly effective, and our subsequent analysis has shown that we over-filtered hypotheses and requested too little data to actually detect many of the relevant threat events. Again, we erred on high precision for events (97%) but had poor recall (57%). Insufficient time prior to the evaluation to test out the integrated

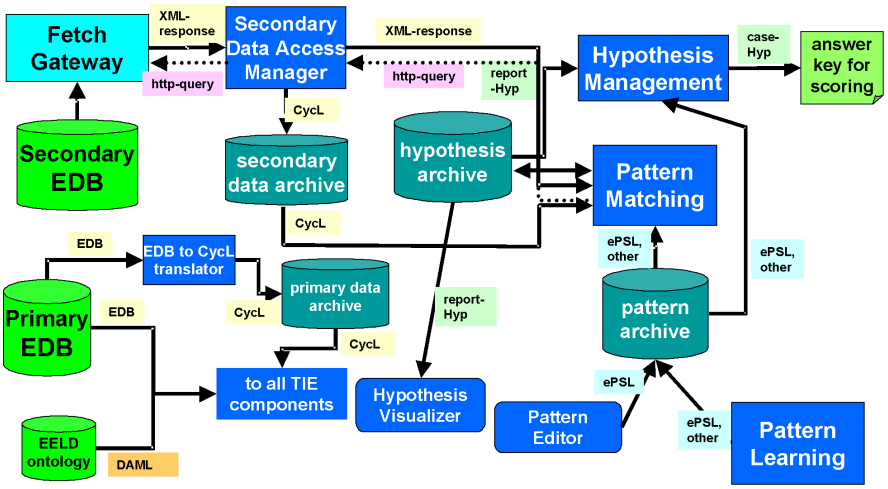


Figure 20: oddTIE architecture

architecture led to this situation. Post-evaluation performance boundary testing demonstrated that loosening the secondary data access restraints significantly improved recall, though naturally precision decreased as a consequence.

Another consequence of the compressed timescales for testing was that multiple passes through the architecture, utilizing SRI's process control infrastructure, were not attempted. It is felt that multiple passes would have significantly improved results.

One surprising result was how little value PL added to the results. This occurred partly because pattern learning was able to identify useful subpatterns in the evaluation data set answer keys, but this did not extend to discovering very useful patterns in the actual data sets themselves. The issue of whether this was a feature of the technologies involved or a characteristic of the evaluation data sets used is still unresolved.

6 Control

LAW is a mixed-initiative, multi-user system. The objective is to allow the analyst to work effectively within the context of a particular scenario of interest. Our original vision of LAW included a central role for a control component that had the ability to exploit structural, semantic, domain, and user-specified information

within a pattern to control the search process, and would task a variety of link discovery (LD) tools and algorithms, exploiting their strengths and compensating for their weaknesses. Our control vision places an emphasis on enabling alternate modes of operation, supporting a user as he develops a pattern for a new scenario or alerting the user to the results of ongoing recurrent queries. During the course of development, we found that the control technology was less critical for the program's and project's goals than other needs, so LAW's current control component is much simpler than the one we proposed, and the more sophisticated control work remains for the future. Nevertheless, we present here several of the most important issues we have identified in the design of a sophisticated control component for link analysis.

6.1 Research Challenges

The search control problem is combinatorially complex. The complexity will depend upon the size of the pattern, the degree of connectedness of the pattern, and the number of disjunctions within the pattern, to name but a few of the relevant dimensions. Exploiting knowledge about a pattern to direct the search is a possible approach. What knowledge is useful, how we represent it, and finally when we apply it are all outstanding questions. Modeling and understanding the capabilities and performance characteristics of the variety of algorithms and tools available for matching patterns and providing secondary source information is nontrivial. Finally, there exists a complex process management problem with the control flow being driven both by user requests (goal-driven) and by intermediate results or derived requests for additional information, perhaps from a secondary source (data-driven).

The LAW team has focused its effort initially on the first two challenges, and this report describes the approaches taken so far and future directions.

6.2 Search Control: exploiting knowledge about the pattern

Consider the search control problem for one complex pattern. It is possible to use an LD tool, for example the LAW Matcher, to match the complete pattern. However, if the pattern is particularly large or has areas of sparsely connected nodes it may be inefficient or ineffective. LD tools will have different strengths; perhaps one excels at matching temporally rich patterns and another at matching small tightly connected patterns. Finally, the search process may be recursive

either because leaf nodes in a pattern may be expandable, or because a partial result may spawn further information gathering.

These observations have implications in the flow of information during search. For example, the search control component may be faced with a variety of choices:

- To send a whole pattern to one or more tools and compare the results
- To send the pattern to one tool and pass its results in the form of partial matches to a second tool and so on
- To decompose the pattern and send portions to different tools depending on the capability of the tool, and merging results, perhaps recursively

The selection of the appropriate tool is discussed in Section 6.3. Here we discuss technologies necessary to facilitate these control options.

6.3 Embedded Control Information

Our initial work in this area has been to allow the user to specify node and link necessity or desirability to guide an individual matcher's algorithm. We will extend this notion to allow a user to tag hierarchical subpatterns or clusters with information about its importance and thus value as an alert, or the sequence in which the pattern should be matched. For example, portions of a pattern may be seen as early indicators for the more complex scenario. This work is supported by our development of a pattern formalism that supports hierarchy and disjunction.

6.4 Strategies to Exploit Structure of a Pattern

The next step is to exploit the hierarchical nature of the pattern formalism automatically, and to identify subpattern types that can be paired with LD tools that can match them effectively. Similarly, the automated clustering of patterns may eventually prove useful. Clusters of tightly connected graphs can be matched, and then specialized tools could be used to search for the connections between the resulting disjointed hypotheses.

The LAW approach combines the highly reactive techniques employed in intelligent process control domains with the more accountable and systematic approach to business management provided by the field of workflow management systems. Structured management of search strategies can provide benefits in several ways. Articulation of explicit search strategies can help to standardize

methodology, thus reducing the likelihood of introduced errors. Explicitly represented search strategies can be tuned to improve efficiency and effectiveness. Automated tools for process management, and thus search control, hold promise for deeper insight into current and planned operations, through the provision of timely updates on progress and status. Such enriched understanding of operating processes will lead to better-informed and more principled decision making by analysts.

At the heart of the SRI controller lies a library of templates that encode explicit strategies for controlling the matching of patterns over data sources, [27]. These templates are represented as explicit sets of tasks or activities to be undertaken, or more abstractly as collections of constraints on allowed activity. This work is built on reactive control and process management technologies [5, 28, 29].

Although much of this work has still to be implemented, we now have a pattern representation format to support these ideas and the underlying process management tools in place. Initial experiments will allow the information professional to specify clusters, hierarchical subpatterns, and semantic control strategies to drive the search.

6.5 Tasking

The tasking of LD tools requires two main technical components: (1) the modelling and representation of capability, and (2) the mechanisms to support Web-based tasking.

We apply work on matching the requirements of some service with the tools to provide that service [12, 1]. LAW takes a semantic-based approach to the provision of Web services and tasking of those services.

> "..software agents should be able to discover, invoke, compose and monitor Web resources offering particular services and having particular properties." [2]

In order to take this approach it is necessary to understand and model the capabilities and properties of a particular tool. LAW provides the semantic tasking necessary to support this approach. As LD tools become available we will model the individual services they provide by using a methodology developed for the workflow management system SWIM [5]. The characteristics to be modeled include the service itself, data over which the service is applicable, strengths and weaknesses according to pattern structure and pattern semantics, performance, and response time.

It is crucial that any system to coordinate and control multiple pattern matching for link discovery be flexible and reactive, and provide the accountability and proactive aids to decision making required by the analysts. The current implementation supports the underlying process control. A user can specify a query to match a pattern and the controller will select and task an appropriate tool to perform the match. LAW can handle multiple queries and multiple users, and it can initiate repetitive queries and manage the hypothesized results.

A Appendix: Metrics for Approximate Pattern Matching

A.1 Introduction to Theory

A.1.1 Approximate Pattern Matching

The complex patterns considered by the matching techniques discussed in this report may be regarded, from a logical perspective, as axioms that specify a *theory* while the objects being sought are *models* of that theory. Informally, the database may be thought of as a collection of objects that are linked by various predefined relations. At a more formal level, facts describing the existence of these objects and their relationships are expressed as the conjunction of the members of a set of instantiated logical predicates such as

`Person(`*Person-1*`),`
`Event(`*Event-3*`),`
`Participated(`*Person-1*`,` *Event-3*`),`
`Employment(`*Person-1*`,` *Company-2*`,` *Position-9*`).`

Correspondingly, patterns may also be conceived in terms of logic constructs, requiring the existence within the database of certain instances of objects and that of links, or relationships, between them. Typically, a pattern will correspond to a closed first-order-logic expression, such as

$$\begin{aligned} \exists\, x, y, z, \quad \ldots \quad & \texttt{Person}(x) \wedge \texttt{Person}(y) \wedge \texttt{Event}(z) \wedge \ldots \\ \ldots \quad & \wedge \texttt{Participated}(x, y) \wedge \texttt{Participated}(x, z) \wedge \ldots \end{aligned}$$

From such a logical perspective, the pattern-matching problem may be regarded as that of finding a correspondence between the variables $x, y, z, \ldots$ and selected database objects (i.e., *variable bindings*) such that the resulting instantiated pattern predicates are, indeed, among the assertions contained in the database. We may also think of the specification of this correspondence between variables and objects as a constructive proof that the database implies the pattern. This perspective is the basis for a number of logical programming languages and of logical approaches to database representation and manipulation [13].

In our treatment of patterns, we generalize this notion of pattern by regarding them as specification of *elastic constraints* on potential models. This type of specification, which is familiar in information-retrieval applications, permits ranking

instances of data structures by their degree of matching with the ideal conditions. For example, a requirement to match the pattern "*Person* P is a *Southern European* who *breeds attack dogs*" might be matched, albeit not perfectly, by an object of the type `Person` who was born in *Central France* (which is close to and overlaps *Southern Europe*) and who *keeps* (but it is unclear whether or not he *breeds*) *wolves*.

In this extended view, patterns do not express strict requirements that are either met or not met. Rather, patterns should be regarded as procedures that rank the adequacy of alternative variable-to-object assignments as potential solutions of a database-retrieval problem. Correspondingly, the values of properties of objects in databases (e.g., *Southern European*) and the nature of the properties themselves (e.g., *breeds*), should be regarded as elastic descriptions that may be met to various degrees. Each possible instantiation matches the pattern to some degree, expressed by a number between 0 and 1 that measures the extent to which such an instance matches the pattern specification. Pattern instances that strictly match, in the logical sense, the pattern specifications have a degree of matching equal to 1, while semantically unrelated instantiations—in a sense to be formalized below—have a degree of matching equal to zero.

Patterns may be regarded, therefore, as mechanisms to measure the *distance*, *similarity*, or *resemblance* of potential solutions of a matching problem to a prototypical set of *ideal* or *perfect* matches. This similarity function, which reflects the semantics of the specific problem being considered, is the basis for the definition of numerical measures of *degree of matching*. This conceptualization of the pattern-matching problem suggests that the pattern-matching problem may be treated as a generalized logical program, that is, as a procedure to search a space of potential solutions and to rank their suitability [36, 18], which extends the theorem-proving approaches of logical programming [13].

In our subsequent discussion, we will define a degree-of-matching function in terms of the *degree of admissibility*, or *adequacy*, of the modifications required to transform a database into a modified counterpart that strictly matches the pattern. Such a best match may be informally described as having the largest admissibility value (i.e., lower transformation cost) among all transformations leading to transformed databases that match the pattern from a classical-logic viewpoint. Database transformations are defined as the composition of a sequence of certain *basic edit operations*. Each edit operation is associated with a numerical value gauging its admissibility. The admissibility of a particular transformation is then defined as a function of the admissibility of its component edits.

The metrics introduced below determine the extent to which a database matches the specifications of a pattern, and are based on semantics provided by knowledge structures such as ontologies. These metrics are intended to be employed in connection with graph-based approaches [41, 8, 48] to database representation. Graph-editing techniques provide a useful framework to describe a variety of complex objects while permitting their comparison in terms of the extent (or cost) of the modifications that are required to transform a graph-based representation of one object into another. These techniques have considerable generality and may be applied to a wide variety of problems. In each application, however, it is necessary that the functions employed to estimate the admissibility of graph transformations reflect the particular meaning attached to each editing operation. This section is devoted to the derivation, from the perspective provided by a combination of logical and metric perspectives, of specific admissibility measures, called *database-editing metrics*, applicable to pattern matching in databases.

A.1.2 Abstract Data Representations

To establish a rational foundation for the definition of semantic measures of similarity, cost, or admissibility, we describe first a representation of the content of various databases—expressed through commonly used data-modeling techniques—in terms of a common description framework based on the notion of *triple*. Each database may be represented either in graph-based terms or as a collection of triples and, correspondingly, graph-based editing operations may be regarded as the modification of some of those triples. This modification leads to a transformed database with a new meaning. The metrics capture, by means of a number between 0 and 1, relevant differences in meaning between the original and transformed database.

We start our discussion by considering several current data modeling approaches and their major conceptual tools. In particular, we describe how the underlying data representation structures of those approaches may be themselves modeled, at a more primitive level, as the conjunction of triples that assert that a *property* or *attribute* of an object has a certain *value*, or that an object has a specific *relationship* to another object. For example, a triple such as

(Age, U33, 25yrs),

indicates that the value of the property Age of the object U33 is "25yrs" while the triple

```
(Son, U33, U323),
```

indicates that the object `U33` has the directed relationship `Son` to the object `U323`. While formally equivalent from a purely logical point of view, these two types of triples, called *attribute predicates* and *object predicates*, respectively, require a somewhat different computational treatment, which is discussed in Section A.2.2.

We proceed now to review the key elements of a number of major approaches to data modeling. We show that it is always possible to represent data models produced by application of these approaches as a set of triples that specify values of properties of objects and relations. Basic database editing operations are transformations, having different degrees of admissibility, that add, delete, or modify these triples.

Although our approach has been developed to deal with databases containing imprecise, uncertain, or vague information, our discussion will first focus—mainly to facilitate understanding—on databases where all asserted properties and relations are precisely defined, that is, each property value is a well-defined element of a value domain (e.g., a number, a string) and each relation—itself belonging to a set of ground predicates—relates two unambiguously specified objects. In this initial treatment. it will be possible, nonetheless, for the information to be incomplete, in the sense that the value of an attribute of an object may not be specified as when, for example, the age of a person is unknown. It will also be possible that, in a similar fashion, certain instances of relationships between objects may not be specified as, for example, when it is not known who is the `Father` of object `P12` of type `Person`. We will not consider now, however, situations where there is partial information about the value of some attribute (e.g., "the Age is more than 20 years"), when there is ambiguity as to the nature of objects linked by a relationship (e.g., "the father of `P12` is either `P23` or `P34`"), or when the relation itself is ambiguous (e.g., "`P12` is either the `brother` or the `cousin` of `P99`").

A.1.3 Data-modeling Approaches

We will examine several data-modeling approaches identifying abstract mathematical and logical structures that capture the essential aspects of linked data structures that are germane to the pattern-matching problem. The purpose of this exercise is to show that data models constructed with these computational tools might also be represented as collections of triples that describe the values of at-

tributes of specific database objects.[6]

Relational models The relational database model [16] is perhaps the formulation that is most readily amenable to conceptual abstraction by means of logical and mathematical structures. In spite of its simplicity, however, some details of the underlying representation—such as the concept of *key*—are often the source of confusion.

The relational model is based on the representation of various objects by tables listing the values of the properties or attributes of objects. For example, a relation of the type `Transaction` may be represented by a table of event-instances describing the event characteristics:

`Date`	`Paid by`	`Payee`	`Transaction Type`	`Amount`
October 1, 1888	David Copperfield	Oliver Twist	Check	$500.00
September 23, 1786	Robinson Crusoe	V. Friday	Cash	$45.56
September 6, 1830	C. Ahab	M. Dick	Stock Transfer	$2100.00

Other tables are employed, as is well known, to describe attributes of other objects such as, in the example above, persons and transaction types. The resulting collection of interrelated tables provides required descriptions of various objects and entities and their relations.

Among alternative formulations, the relational model affords the best compromise between the requirements implicit in the representation of complex links between real-world entities and the need to rely on a simple abstract structure capable of being captured by labeled graphs. In particular, we may note that relational tables, beyond identifying values of properties and attributes, identify, as part of the database schema, the properties and attributes themselves (e.g., `Date`, `Payee`). Furthermore, it is clear that individual entries may point to instances of other represented objects (e.g., "C. Ahab") or to particular elements of certain primitive sets of *values* (e.g., a particular date).

The relational model, however, was conceived in the context of classical applications where the characteristics of entities (i.e., specific "rows" in a relational

[6]The representation of data by triples and the derivation of equivalent graph representations are well-known [3, 17]. The nature of the calculus of binary relations, from a historical perspective, has been discussed by Pratt [31].

table) are uniquely identified, within the data model, by the values of a subset of some attributes (i.e., the *key*),[7] In applications where the attributes are not precisely known, the information contained in the key may not be sufficient to uniquely identify the relational entry by means of the functional mapping that is implicit in the key-to-attribute mapping. For example, the name of a person may be only partially known, and other identification characteristics might be unknown or unreliable. Furthermore, it is possible that two different entries in a relational table correspond to the same object in the real world (e.g., a person is represented using two entries with different aliases or identity documents). Conversely, two identical imprecise entries might correspond to different objects in the real world.

To be able to refer, under such conditions, to each specific tabular entry (henceforth called a *relationship*) we need to be able to uniquely identify it. In our previous example, after identifying specific "person" entries in the relation **Persons** describing attributes of individuals (some of which may describe the same individual) and providing a unique identifier to each relationship in the relation **Transaction**, we have the new relation[8]:

Trans-ID	Date	Paid by	Payee	Transaction Type	Amount
T-1	Oct. 1, 1888	P-5	P-9	Check	$500.00
T-2	Sep. 23, 1786	P-78	P-97	Cash	$45.56
T-3	Sep. 6, 1830	P-112	P-323	Stock Transfer	$2100.00

From this representation, we may readily describe the relationships of the data model by triples such as

```
(Payee, T-2, P-97),
(Amount, T-2, $45.56),
```

which identifies how links, such as `Payee` or `Amount`, relate a specific instance of an object (e.g., `T-2`) with an instance of another object (e.g., `P-97`), or with

[7] Our discussion of the relational model and other data-modeling approaches has been simplified, focusing solely on major issues related to graph-based modeling of data structures.

[8] The identifiers are prefixed by a letter intended to identify the type of object being identified or linked. This choice was motivated by the desire to preserve some understandability of the example. In general, identifiers are introduced to uniquely identify a relationship between *data entities*.

a value (e.g., \$45.56, respectively). This is a binary relation between object-instances or between object-instances and values of the type identified by the name of a particular attribute or property.

A.1.4 Logic-based Representations

Approaches based on logical representations such as those inspired in the LISP and PROLOG programming languages [26, 13] are based on symbolic representations of data, such as $p(O_1)$, $q(O_2, O_3)$, $r(O_{11}, O_{12}, \ldots, O_{1n})$, interpreted as knowledge about the properties of various instances of objects.

Although there have been important extensions of these languages that permit explicit representation of semantic knowledge [4] (e.g., the meaning of the arguments in n-ary predicates), in general, the interpretation of symbolic expressions is usually not made clear or is hidden in the program itself.

There is a simple conceptual mapping, however, between relational entries and collections of logical representations of instantiated predicates. For example, a fact represented as $r(O_{11}, O_{12}, \ldots, O_{1n})$ may be represented in a relational setting as an entry $O_{11}, O_{12}, \ldots, O_{1n})$ in the relation r. Since there may be multiple instances of known validity of the relation r and since the values of the related variables may not be precisely known, it is necessary again to provide a unique system identifier permitting differentiation between two known instances

$$r(O_{11}, O_{12}, \ldots, O_{1n}) \quad \text{and} \quad r(O_{21}, O_{22}, \ldots, O_{2n}) \,,$$

of validity of the predicate r, by assigning a unique identifier r_{ID} to each instance.

From such a conceptual mapping, it is easy to see that symbolic representations such as $q(O_2, O_3)$ may be alternatively represented as triples of the form

$$(\text{Arg}_1, q_{\text{ID}}, O_2), \quad (\text{Arg}_2, q_{\text{ID}}, O_3)$$

where q_{ID} identifies the particular instantiation of the predicate q being represented using knowledge about its properties or attributes Arg_i.[9]

Frames, logical records, and similar structures Data-based structures relying on representation of properties of entities by the values of their properties are

[9]The semantics of both relational identifiers and that of the related variables (or arguments) is sometimes made explicit through structures that play the role of a database schema.

conceptually similar to instances of relations.[10]

Logical records describe the values and attributes of data objects while identifying related object instances and the nature of such relationships. These descriptions (i.e., *fields, slots*), play the same role as column headers in relational tables. From this observation, it should be clear that any such structure may be represented as a collection of triples of the type `(Field, RecordID, Value)` where `RecordID` is the identifier of a particular record (or frame), `Field` identifies a particular field or slot, and `Value` is the value of the corresponding property or attribute associated with `Field`.

Entity-relation models Models based on the entity-relationship (ER) model of Chen [11] rely on the two basic structures defining properties of real-world entities (i.e., entities) and their relations to other entities. In either case it is clear that specification of values of properties or attributes of entities

$$(\texttt{Attribute}, \texttt{EntityID}, \texttt{Value}),$$

or statements of relations between instances of entities

$$(\texttt{Role}, \texttt{EntityID}_1, \texttt{EntityID}_2),$$

where `Role` identifies the nature of a link in a n-ary relation, by means of triples provide a description of entities, their links, and the nature of their links that fully captures the information described by ER structures.

A.1.5 Graph Editing

We now briefly recall the basic characteristics of the graph editing approach of LAW, seeking to relate it with the database metrics.

The conceptual structures underlying most data representation methods as well as their instantiation by population with specific objects and relationships lend themselves to graph-based representations [42]. Graphs provide an effective way to visualize database structures and contents while facilitating the specification of patterns. From a pattern-matching viewpoint, graph-based structures also provide insights into the nature of the sequences of editing operations leading from a given database to a transformed prototype of the pattern (i.e., it meets the

[10] This comment reflects solely concern with graph-based representation of basic links between data objects. *Frames* and similar formalisms also provide tools for the specification of complex constraints beyond the current scope of this report.

pattern constraints as a classical-logic restriction). The extent of the modifications required to attain such a transformation may then be employed as the basis to measure the degree of matching between original and prototype.

Graph-editing approaches rely, however, on notions of cost or adequacy of graph transformations, which are highly dependent on the meaning of such transformations in the context of the problem under consideration. In the case of pattern matching in large databases, each basic editing operation, such as an edge addition, corresponds to one or more basic database operations, that is, addition, modification, or deletion of triples in the triple-based representation of the database. Each such triple-based operation changes the meaning of the database to some extent that is commensurate with the nature of the modification. Correspondingly, any measure of admissibility must reflect the importance of the modifications to the information contained in the original database.

We derive rational bases for the measurement of the admissibility of graph-editing operations in pattern matching by considering measures of semantic similarity between original and transformed databases. The latter measures are derived from knowledge structures, such as ontologies, that permit determination of the extent of resemblance between objects from various viewpoints.

We start our discussion by recalling, for convenience, the basic concepts underlying graph editing without regard for the nature of the problem being addressed. Our subsequent discussion, however, focuses on the derivation of measures of cost and admissibility that are germane to pattern matching.

Graph transformations We sketch now the essentials of a theory of graph-editing transformations.

A **graph** is a 4-tuple $G = (V, E, \mu, \nu)$ where

(i) V is a finite set of *vertices* or *nodes*.

(ii) E, the set of *edges*, is a subset of $V \times V$.

(iii) $\mu : V \to \mathcal{L}_V$ is a function assigning labels to the vertices.

(iv) $\nu : E \to \mathcal{L}_E$ is a function assigning labels to the edges.

A **graph mapping** between two graphs $G = (V, E, \mu, \nu)$ and $G' = (V', E', \mu', \nu')$ is a pair $M = (\psi, \varepsilon)$ where

(i) $\psi : V_0 \to V_0'$ is a one-to-one mapping between a subset of vertices V_0 of V and a subset of vertices V_0' of V'.

(ii) $\varepsilon : E_0 \rightarrow E'_0$ is a one-to-one mapping between a subset E_0 of the edges E of V and a subset E'_0 of the edges E' of V'.

such that if two edges are mapped by ε, then the nodes connected by those edges are mapped by ψ.

In graph-editing approaches, graph transformations are constructed as the composition of a finite number of basic graph-editing operations. Graph-editing operations include mappings between graphs that result in additions, deletions, and replacements of individual nodes and edges.
A **transformation** T between two graphs G and G' is the composition of a sequence of edits $(\rightarrow_0, \ldots, \rightarrow_m)$.

Degree of matching The notion of admissibility, of a transformation is intended to capture, by means of a suitable metric that reflects the semantics of the domain being modeled, the significance of its effects. Transformations involving editing operations that highly distort the nature of the original object are assigned a low admissibility value, while those involving minor modifications have a high admissibility value. An admissibility measure Ad is a function that assigns a number between 0 and 1 to every transformation T between graphs.

A transformation may be thought of as a path in graph-space connecting a graph with its transformed version with each edit operation being a step on that path. From such a perspective, it makes sense to define the admissibility of a transformation in terms of some function that aggregates the admissibility of each of its component edits.

Several functions, known as *triangular*, or *T-norms* [35], can be shown to have desirable properties that accomplish such aggregation:

Definition 1: The **admissibility of a transformation** is the aggregation of the admissibility of its edits by means of a triangular norm $\circledast$, that is,

$$\mathsf{Ad}(T) = \circledast_{e \in T} \mathsf{Ad}(e)\,.$$

Several transformations may be consistent with a mapping between two graphs. It makes sense, therefore, to measure the admissibility of the mapping between two graphs in terms of the transformation having maximum admissibility:

The **degree of matching** between two graphs G and G' is the admissibility of the transformation T changing G into G' that has the largest admissibility value among all such transformations.

A.1.6 Similarity

The notion of *similarity* or *resemblance* is central to the approach presented in this report. Similarity measures provide the bases for the determination of the *admissibility* of certain transformations of a database into another that meets the logical conditions expressed by a pattern. The notion of admissibility, which may be thought of as being dual to the concept of cost,[11] is introduced to indicate that certain database modifications are more permissible than others. The basic idea is that transformations resulting in a similar database are more admissible (i.e., less costly) than those resulting in substantial difference between the original and transformed data. The *degree of admissibility* of a database transformation is defined in terms of the numerical measures of similarity, which are themselves the counterpart of the notion of *distance*. Similarity measures, mapping pairs of objects into a numeric value between 0 and 1, provide a desirable foundation for the development of a rational theory of database editing, not only because they are related to notions of cost, utility, and admissibilty but also because they provide the basis to extend classical logical relations of inference to approximate, multivalued, counterparts [33].

A.1.7 Similarity Measures

Similarity functions are measures of *indistinguishability* that may be thought of as being the dual of the notion of bounded distance (i.e., a distance function taking values between 0 and 1). A similarity measure assigns a value between 0 and 1 to every pair of objects o and o' in some space X. Typically, similarity measures d may be obtained from knowledge of distance functions d by simple relations [35] such as $S = 1 - d$. The similarity of an object o to itself is always equal to 1 (corresponding to a distance equal to zero), while the minimum possible value for a similarity function is 0.

The advantage of the use of similarities in lieu of distances lies on their advantages as the foundations of logics of utility [36], which provide the bases to combine on a rational basis, measures defining utility, admissibility, and cost from various perspectives. A detailed discussion of the notion of similarity is beyond the scope of this report. We limit ourselves to pointing out the important properties of similarity (or generalized *equivalence*) functions:

[11] The numerical degree of admissibility of a database transformation is the complement of the notion of cost in the sense that low costs correspond to high admissibility values while high costs correspond to low admissibility values.

Reflexivity: $S(x, x) = 1$, for all x in X,

Symmetry: $S(x, y) = S(y, x)$, for all $x,\ y$ in X.

Transitivity: $S(x, y) \geq S(x, z) \circledast S(y, z)$, for all $x,\ y,\ z$ in X, where $\circledast$ is one of a family of numerical binary operators called *triangular norms* or T-norms.

The transitivity property is of particular importance as it extends the transitive properties of classical deductive methods (i.e., the transitvity of the inferential procedure known as *modus ponens*) into multivalued-logic schemes capable of modeling numeric measures of implication and compatibility [36].

Similarity and utility The metric notion of similarity is closely related to utilitarian notions such as *cost, admissibility*, and *utility* [34].

Utility functions defined on a domain $\mathsf{Dom}_{\mathcal{O}}$ assign to every object O of that domain a number $u(O)$ between 0 and 1, which measures the extent to which the situation represented by that object is good or desirable.

In robotics problems, for example, the position of a mobile autonomous robot may be said to be good because it is "not in danger of hitting an obstacle," or "because it is in communication with other robot" [39]. In a control-systems or decision problem, different actions may be good or bad, from various viewpoints, depending on their outcome (i.e., the state of the system being regulated).

In pattern-matching problems, utility functions provide a convenient procedure to define generalized predicates that measure the extent to which an object meets some elastic constraint from the perspective of the usefulness of that object as part of the instantiation of a pattern. For example, if a pattern requires that the amount of a `Transaction` should be "large," then the definition of the term "large" provides a procedure to measure the adequacy of any possible amount as part of a pattern requiring that an object of the type `Transaction` has a large value. In the case of predicates involving various arguments, such as `NearlyOneHourLater`(*time*$_1$, *time*$_2$), the corresponding utility function gauges the adequacy of a pair of time values as arguments satisfying the relation `NearlyOneHourLater`.[12]

Utility functions provide, therefore, a convenient way to rank the desirability of events, situations, and objects in a numerical scale. As such, they have been a central concept in modern decision theory [32].

[12]This characterization is the basis for the utility-based interpretation of fuzzy sets and fuzzy relations [36].

If several utility functions $u_1, u_2, \ldots, u_n$ are considered when comparing pairs of objects in a domain O, then a similarity function may be defined by composition of criteria defined for each similarity [47]:

$$S(O, O') = \min_i [\, |u_i(O) \oslash u_i(O')| \,] \,,$$

where $\oslash$ is the pseudoinverse of the triangular norm $\circledast$, and where $|a \oslash b|$ stands for $\min(a \oslash b, b \oslash a)$. Basically, the above result, known as *Valverde's Representation Theorem* states that two objects, events, or situations are similar if, from every important viewpoint, they have similar utility values.

The notion of utility is also the basis for certain multivalued logics [18] and, more important, for the derivation of proof procedures that are directly applicable to problems such as pattern matching (i.e., as this problem is equivalent to proving that the data implies the pattern). Recent results indicate that these logics are closely related to the similarity-based logics underlying our approach to pattern matching [36]. Conversely, similarity measures may be employed to derive measures of cost and admissibility.

Numerical measures of admissibility We discuss now approaches to the definition of similarity and admissibility measures in the context of pattern-matching problems.

Semantic distance, similarity, and ontologies Our simplest scheme for numerical assessment of the admissibility of edit operations is based on the similarity measures between leaves of an ontological directed acyclical graph (DAG) that measure the extent to which leaf nodes share ancestors in the ontology [36]. This scheme, first proposed by Ruspini and Lowrance in the context of the development of the SRI's SEAS system [38], is also similar, in spirit, to approaches to defining semantic distance between concepts on the basis of the knowledge provided by a generalized thesaurus [7].

These ontology-based measures of similarity permit only, however, gauging the resemblance between different types of unlinked objects. Pattern-matching problems, however, require the consideration of similarity between complex linked structures where the similarity between two such structures depends on the nature of the links and that of the attributes of the related objects. To address this problem, we discuss mechanisms for the derivation of complex similarity measures in terms of simpler constructs.

In our discussions, we will, mainly for the sake of simplicity, assume that every node in an ontology is relevant to the measurement of similarity between classes of objects. In general, however, the measurement of similarity between types is made on the basis of a connected subgraph of the ontology as many properties might be irrelevant to certain types of comparison (e.g., certain items of furniture and cows both have four legs but this shared property is generally irrelevant to the characterization of similarity between these rather different objects).

Being closely related to the notion of distance, several measures of similarity have been proposed to address problems ranging in nature from sociology and psychology to pattern recognition [41, 6]. Our approach to the description of similarities between basic model objects exploits the availability of domain knowledge in the form of ontologies of various object domains and ontologies of relations.

Ontologies provide semantic bases for the definition of notions of distance, resemblance, and similarity between concepts. In most applications of knowledge-based concepts, objects belong to certain classes, or *types*. Ontologies, through class subsumption structures, corresponding to a DAG, permit the definition of a distance function between elements of the ontology, as done by Wolverton [48], on the basis of the length of the paths linking two elements of the ontology.

While this type of approach provides a simple mechanism to gauge conceptual proximity on the basis of domain semantics provided by ontologies, the identification of a framework to measure the extent by which a representation of data objects and their relations match a reference pattern requires development of a more sophisticated methodology. Several considerations support this conclusion:

- The process of pattern matching is inherently *nonsymmetric*, since the editing operations required to transform one pattern into another are different from those accomplishing the inverse mapping.

  For example, the admissibility of the operation replacing an object of the type `Italian` by another of the type `European`, in order to satisfy a pattern requirement, (identifying a *set* of `Nationalities`) should be 1 (i.e., the associated cost should be 0), as the data is more specific than the requirement expressed by the pattern, since every possible instance of an Italian is an instance of an European. The replacement of a value node of the type `European` by another with a value of `Italian` to match a pattern requirement, should, in general, have a measure of admissibility that is strictly smaller than 1 since it assumes additional knowledge.

- In general, the admissibility of editing operations replacing a value set with

another value set cannot be uniquely determined. For example, computing the costs associated with replacing the label `SouthernEuropean` of a value set node (describing the set where the attribute `Nationality` of an instance of `Person` is known to be) with the label `WesternEuropean` might very well have a potential admissibility of 1 (i.e., a cost of zero), as there are some nationalities (e.g., `Spanish`) in the intersection of the value sets. On the other hand, the different nature and extension of the classes indicates that there may be a *potential cost* (corresponding to an admissibility value strictly smaller than 1) associated with that replacement (e.g., additional information may reveal that the person was Bulgarian). This range of possibilities suggests that, in many cases, the editing cost may be better represented by an *interval of possible values* rather than by a single number.

- The measurement scheme should reflect the extent to which different classes of objects share important common properties. Ontologies, identifying various relations of set inclusion among the classes represented by ontology types, provide a solid foundation for the measurement of resemblance on a semantic basis. Path lengths on the ontological graph, however, do not accurately reflect, in most instances, this semantic knowledge.[13]

We propose next a scheme to measure costs by subintervals of the [0,1] interval of the real line representing, on the basis of knowledge provided by ontologies, the *possible costs* associated with basic editing operations.

Similarity between leaf nodes The relations of set inclusion between nodes in an ontology may be represented in a number of ways by vectors describing whether a node of some type is the ancestor of another node. Ruspini and Lowrance [38] suggested a representation of the j-th node in terms of a vector having the length n of the cardinality of the ontology with the k-th component of that vector, representing whether or not the k-th node is an ancestor of the j-th node, that is,

$$v_k(N_j) = \begin{cases} 1, & \text{if node } k \text{ is an ancestor of node } j. \\ 0, & \text{otherwise} \end{cases}$$

[13] In an ontology of `Animals`, for example, the distance between the classes `Dog` and `Cat` is strictly smaller than that between any particular breed of dog and any particular breed of cat.

On the basis of this representation the degree of similarity between the leaf nodes N_i and N_j may be defined by

$$\hat{S}(\, N_i, N_j \,) = \frac{< v(N_i), v(N_j) > -1}{\sqrt{\left(||v(N_i)||^2 - 1\right)\left(||v(N_j)||^2 - 1\right)}}\,,$$

where $v(N_i)$ and $v(N_j)$ are the just-introduced binary vector representations, of the nodes N_i and N_j.[14]

This similarity measure is a symmetric function taking values between 0 and 1, such that the similarity of a leaf node to itself is always equal to 1.

On the basis of this measure of similarity, it is possible—as will be shown when discussing extensions of the notion of similarity between objects to similarity between classes of objects—to derive interval measures that gauge the *necessary* and *possible* admissibility of graph-editing transformations.

Complex similarity measures While ontologies permit the construction of semantic-based similarity functions between basic objects (e.g., `Countries` by Type of Economy) and between values of attributes (e.g., , `Age`), it is often the case that many of the structures found in a typical pattern-matching problem, involving complex links between primitive objects, may not be amenable to this type of treatment.

Consider, for example, the set of linked objects characterizing an illegal transaction such as `Money Laundering.` Unless this type of object has been subject to some form of ontological characterization, it should be clear that any measure of similarity between objects of this type needs to be based on the similarity of attributes of corresponding objects in each structure.[15]

These complex structures may be characterized, however, through logical expressions, such as

$$\begin{aligned}\texttt{Person}(x) \;\wedge\; & \texttt{Person}(y) \wedge \texttt{Money-Transfer}(z) \wedge \\ \wedge\; & \texttt{Depositor}(x,z) \wedge \texttt{Payee}(x,z) \wedge \ldots \Rightarrow \texttt{Money-Laundering}(x,y,z\ldots)\,,\end{aligned}$$

[14]The definition assumes that the root node is not itself a leaf node.

[15]In this regard, it is important to note that ontologies, if available, already summarize the important distinctions between objects in terms of their significant attributes. The characterization of an object, such as a `Person` in terms of its ontological lineage as opposed to a vector of attributes is a matter of choice. In the case of complex, domain-specific, structures, however, it is reasonable to assume that ontologies will not be readily available, thus requiring a different similarity-measurement approach.

which can be employed as the basis for the definition of a similarity measure between money-laundering events as a function of the similarities between the various basic ground predicates and the logical operators that interconnect them [37].

In general, the computation of these complex measures is straightforward. In some cases, however, as when trying to measure similarity between say, people, by the type of company they keep, application of the above approach results in a definition that depends on other values of the measure being defined, as the similarities between associates depend on the nature of the very people being compared (since human association is a symmetric relation and the people being compared are themselves associates of their associates). While the required measure may be usually derived by iteration,[16] it is important to reduce the complexity of the definition as the underlying computational problem may be intractable. In these cases, common when considering transitive relations, it may be required to limit the extent of the structures being compared to reduce such computations.

Generalized similarity measures In order to determine, on a sound formal basis, the admissibility of each editing operation in terms of the nature of the objects and links involved it is necessary to derive a general metric characterizing the acceptability of the outcome of an editing operation from the viewpoint of the reference pattern being matched.

Ruspini [33, 37] proposed, in the context of studies about interpretations of fuzzy-logic concepts, the measurement of the similarity between two sets by extension to a logical framework of concepts, notably the well-known notion of *Hausdorff distance*, from the theory of metric spaces. The connection between these concepts and utilitarian interpretations of certain *possibilistic* constructs has been studied recently to a considerable extent [36].

The basic construct of this theory is the function

$$\mathbf{I}(A \mid B) = \min_{o \in B} \max_{o' \in A} S(o, o') \,,$$

defined also over pairs of subsets of X, which measures the *degree of inclusion* of B in A with respect to S, that is, the extent of the minimal metric neighborhood of B that encloses A (in the sense of set inclusion).

[16] The definition of similarity in these cases is equivalent in character to the "fixed point" definitions of classical logic. To use another analogy, we may say that the similarity measure is being defined through an implicit equation.

The nonsymmetric metric $\mathbf{I}$ will be useful in the definition of costs of basic editing operations as it measures the extent by which the concept represented by the class A needs to be extended (or "stretched") to encompass that described by the class B. The metric $\mathbf{I}$ is a measure of set inclusion since a value of $\mathbf{I}(A \mid B)$ equal to one indicates that every member of B is also a member of A, that is, that B is included in A.

The dual of the degree of inclusion measure is that of *degree of intersection*, defined by

$$\mathbf{\Pi}(A, B) = \max_{o \in A} \max_{o' \in B} S(o, o') \,.$$

measuring the extent to which either set has to be extended to intersect the other. When this number is 1, then the sets have at least one point in common. On the other hand, a large value of the *possibility measure* $\mathbf{\Pi}(A, B)$ indicates that all points of A are far apart from all points of B.

It is clear from the definitions of $\mathbf{I}$ and of $\mathbf{\Pi}$ that it is always $\mathbf{I}(A \mid B) \leq \mathbf{\Pi}(A, B)$. The numerical interval $[\mathbf{I}(A \mid B), \mathbf{\Pi}(A, B)]$ represents the potential values of the degree of similarity between an object o' in B and its closest object o in A.

The function $\mathbf{I}$ has a number of useful properties that are the foundation of our approach to the estimation of the degree of matching between a pattern and a database in terms of the $\circledast$-aggregation of the admissibilities of a sequence of simpler editing operations. The following properties are of particular importance to pattern matching.
If A, B, and C are subsets of a domain $\mathbf{X}$, then

1. If $B \subseteq A$, then $\mathbf{I}(A \mid B) = 1$,
2. $\mathbf{I}(A \mid B) \geq \mathbf{I}(A \mid C) \circledast \mathbf{I}(C \mid B)$,
3. $\mathbf{I}(A \mid B) = \max_{C} [\mathbf{I}(A \mid C) \circledast \mathbf{I}(C \mid B)]$.

In what follows, we will need to consider situations where single objects are either added or deleted to an existing set A to produce a transformed set A'. In such cases, we will need to compute the degree of implication $\mathbf{I}(A \mid A')$ as the measure of the admissibility of the simple operation changing A into A'. In those cases, it is straightforward to see that

1. If a member o of A is deleted to produce A', that is if $A = A' \cup \{o\}$, then $\mathbf{I}(A \mid A') = 1$.

2. If a member o of $\mathbf{X}$ is added to A to produce A', that is, if $A' = A \cup \{o\}$, then $\mathbf{I}(A \mid A') = \max_{o' \in A} S(o, o')$, where S is the similarity function underlying $\mathbf{I}$.

A.2 Pattern Matching and Data Transformations

We present the basic elements that permit the estimation of the degree of matching between a pattern and a database.

We start our discussion by characterizing the basic data constructs permitting the characterization of a database as a collection of assertions about binary relations between objects or between objects and their properties. On the basis of that characterization, we present a formal definition of a *Database* as a set of instances of binary predicates. The database may be interpreted, in logical terms, as the conjunction of that set of predicate instances.

We proceed then to sketch the general characteristics of our editing approach by means of definitions of the degree of admissibility of sequences of basic database-editing operations and that of degree of matching. Finally, we present results characterizing the degree of admissibility of basic database-editing operations in terms of related similarity functions.

A.2.1 Predicates, Objects, Attributes, and Values

Our brief review of major data modeling approaches in Section A.1.3 showed that their underlying representation mechanisms may be mapped into an equivalent collection of triples describing the nature of a relation between a *pair of objects*, or between an *object* and one of its *attributes* or *properties*.

Following OpenCyc [14] nomenclature, we will refer to these classes of structures as being either *binary object predicates* or *binary attribute predicates*, respectively (or *object predicates* and *attribute predicates*, for short).[17]

Instances of object predicates such as `(isFather, P12, P34)`, sometimes called *relationships* in the database literature, state that the two object instances are related by the specified relation. Instances of attribute predicates, such

[17] It is important to note that the qualifier "binary" correctly describes the arity of logical predicates defining a relation between two objects or the relation between an object and one of its attributes, respectively. The use of the term *triple* is justified, however, as three components need to be specified to unambiguously identify an instance of a binary predicate (i.e., (*predicate-instance object-instance object-instance*) or (*predicate-instance object-instance attribute-value*), respectively).

as (hasAge, P12, 55yrs), specify the value of a property of an object instance. In what follows, and in deference to terminology employed in the graph-theoretical and database literature, we will also refer to specific instances of object predicates and attribute predicates as *links*.

A.2.2 Databases

After presenting the basic constructs underlying our approach and reviewing the important logical, metric, and utilitarian structures that relate them, we are now in a position to make a formal definition of a general data model for the representation of precise data and knowledge.[18]

Abstract Data Model We start our discussion with a characterization of the various knowledge and data components of a data repository.

Definition 2: A data model is a 4-tuple

$$\mathcal{D} = (\,\mathsf{Obj}, \mathsf{Vals}, \mathsf{Pred}, \mathsf{Data}\,)\,,$$

where

(i) *Objects*: Obj is a nonempty set, called the set of *objects*.

It is often the case that objects are structured by means of knowledge structures, such as ontologies, describing set-inclusion and subsumption relations between objects as well as the conditions upon object properties that make possible the differentiation of subclasses (e.g., the property of Animals that makes Vertebrates different from Invertebrates.

(ii) *Values*: Values are nonempty sets in a collection of basic primitive types Vals, such as numbers, strings, or members of predefined discrete sets such as

$$\texttt{BalkanCountries} = \{\,\texttt{Albania}, \texttt{Bosnia}, \ldots, \texttt{Yugoslavia}\,\}$$

that permit specification of the values of properties and attributes of objects.

[18] As previously noted, our model allows representation of *ignorance* about the values of attributes or about the possible existence of relations between database objects.

(iii) *Predicates*: Pred is a nonempty set, called the set of *predicates*, or the set of *links*.

Links may also be thought of as being relations defined between objects in some domain and the set of possible values of some *property* or *attribute*. Members (in the set-theoretic sense) of such a relation are predicate, or link, *instances*, being sometimes also called *relationships*. As we have already pointed out, there is considerable latitude in data modeling as to what constitutes an entity (related to values of its attributes) and what is a relation (related to other objects). In the context of our model, predicates are different from objects in that the latter are instantiated and identified as a specific data object of a certain class (its domain), while the former are generic and are not instantiated.

Since predicates are typically members of some domain that is structured by knowledge constructs such as ontologies, we will assume that, in general, there exists a similarity function $\mathsf{Sim}^{L}_{\mathcal{P}}$ defined between pairs of predicates.[19]

(iv) *Predicate Classes and Instances*: As discussed earlier, we will need to consider two classes of predicates, called *binary object predicates* and *binary attributes predicates*, respectively.

Instances of object predicates expressed by triples such as

```
(predicate, object-instance, object-instances)
```

state that two object instances are related by a specific object predicate, as, for example, in `(isAncestor, P123, P234)`.

Instances of attribute predicates, expressed by triples such as

```
(predicate, object-instance, attribute-value)
```

specify the value (that is, a unique element of some primitive set in the collection Vals) of an attribute of the specified object instance, as, for example, in `(hasCityAddress, P543, ''Palo Alto'')`.

[19]This function defined between predicates should not be confused with similar metrics, discussed below, between pairs of predicate *instances*.

We will assume that the sets $\mathcal{P}_O$ and $\mathcal{P}_A$ of predicate instances have a metric structure defined by means of similarity functions $\mathsf{Sim}^O_{\mathcal{P}}$ and $\mathsf{Sim}^A_{\mathcal{P}}$, respectively. We discuss below an approach to the definition of such similarity measures.

(v) *Data*: The data Data is a tuple $(\mathsf{Data}_{\mathcal{O}}, \mathsf{Data}_{\mathcal{A}})$, where $\mathsf{Data}_{\mathcal{O}}$ is a set of object-predicate instances and $\mathsf{Data}_{\mathcal{A}}$ is a *nonempty* set of attribute-predicate instances.

This formal definition simply says that the database is a collection of triples relating objects and a nonempty collection of triples specifying the values of properties and attributes of objects. We require the latter collection to be nonempty to assure that the information represented in the database is grounded on objects that are described, to some extent, by their properties.

A.2.3 Similarities between Predicate Instances

The definition of a metric structure in the sets $\mathcal{P}_O$ and $\mathcal{P}_A$ of predicate instances is an essential element of our approach since it permits definition, on a rational semantic basis, of the admissibility functions $\mathsf{Ad}^O_{\mathcal{P}}$ and $\mathsf{Ad}^A_{\mathcal{P}}$ that measure the adequacy of basic database editing transformations.

We have shown that knowledge structures such as ontologies permit the definition of similarity measures in various domains. Our approach will not place any constraints on the nature of the similarity measures $\mathsf{Sim}^O_{\mathcal{P}}$ and $\mathsf{Sim}^A_{\mathcal{P}}$ between predicate instances other than assuming that there are similarity functions defined within object-instances (on the basis of their relationships and values of their properties) and between specific property values.[20]

Among the many possibilities open for such definitions, however, there is a simple choice that defines similarity between triples as the composition of the similarities between each of the triple components. A straightforward definition of the similarity between triples in terms of simpler measures defined over their three components, however, is not possible as, usually, these measures depend on the nature of the predicate being considered. A similarity function defined over the set of integer numbers may be employed to compare person ages (measured

[20]To keep the scheme as general as possible, we will assume that it is possible to define a function between any two members of some set in the collection Vals (e.g., between a number and a string). In practice, however, many of these comparisons will be meaningless and the correspondingly similarity values will be equal to zero.

in years), while a different metric may be required to compare their weights (expressed in kilograms). Although both similarity functions compare elements of the same primitive set (i.e., integers), it is clear that their definition is strongly dependent on the nature of the predicates involved (i.e., `hasAge` and `hasWeight`, respectively).

To arrive at a simple suitable formulation that properly addresses this problem, consider first the problem of defining the similarity of two attribute-predicate instances (l, o, v) and (l, o', v') having the same first component, that is, the same attribute predicate.

Assuming that, for every attribute predicate l, there exists

1. a similarity function $\mathsf{Sim}^l_{\mathcal{O}}$ defined between pairs of objects in Obj (i.e., a way to compare possible replacements of the second component of the tuple)

2. a similarity function $\mathsf{Sim}^l_{\mathcal{A}}$ defined between pairs of attributes each lying in some (usually, but not necessarily, the same) primitive value set contained in the collection Vals (i.e., a way to compare possible replacements of the third component of the tuple)

we can now define a similarity function in the set $\mathcal{P}^l_A = \{(l, o, v)\colon o$ is an object, , v is a value$\}$

$$\mathsf{Sim}^A_{\mathcal{P}}(\,(l, o, v),\ (l, o', v'))\, = \mathsf{Sim}^l_{\mathcal{O}}(o, o') \circledast \mathsf{Sim}^l_{\mathcal{A}}(v, v')\,,$$

where $\circledast$ is a T-norm such that $\mathsf{Sim}^l_{\mathcal{O}}$, and $\mathsf{Sim}^l_{\mathcal{A}}$ are $\circledast$-transitive for all links l in Pred.

Trying to extend this definition to the case where it is necessary to measure the resemblance between attribute-predicate instances (l, o, v) and $(l'o', v')$, involving different predicates l and l', we resort to the $\circledast$-transitive similarity function $\mathsf{Sim}^L_{\mathcal{P}}$, introduced earlier, which is defined between pairs of attribute-predicates in Pred. Although this function provides the bases for the required extension, the dependence of metrics measuring resemblance between objects and between attribute values on the nature of the attribute predicate l demands that, when comparing (l, o, v) and $(l'o', v')$, we define whether we employ metrics that measure such similitudes either from the viewpoint of l or from that of l'.

While several choices are possible, our preferred approach is to require that, in order for (l, o, v) to be similar to $(l'o', v')$, the following conditions must be met:

1. The predicates l and l' should be similar.

2. The objects o and o' should be similar from the viewpoint of the predicate l *and* from the viewpoint of the predicate l'.

3. The attribute values v and v' should be similar from the viewpoint of the predicate l *and* from the viewpoint of the predicate l'.

To meet the desired requirement that the objects o and o' be similar from the viewpoint of the predicates l and l' (similarly with the attribute values v and v'), we need to define a function that combines the numeric distinctions made both by $\mathsf{Sim}^{l}_{\mathcal{O}}$ and $\mathsf{Sim}^{l'}_{\mathcal{O}}$ (similarly, $\mathsf{Sim}^{l}_{\mathcal{A}}$ and $\mathsf{Sim}^{l'}_{\mathcal{A}}$). The following result, stated without proof, permits to define the required similarity function:
Theorem: *Let S_1 and S_2 be $\circledast$-transitive similarity functions between objects in a domain* **X**. *The function $S_{12} = \min(S_1, S_2)$, , is the largest $\circledast$-transitive similarity function such that $S_{12} \leq S_1$ and $S_{12} \leq S_2$.*
This result allows us to define a similarity function that measures the required distinctions from the criteria imposed by both attribute predicates l and l':

$$\begin{aligned} \mathsf{Sim}^{(l,l')}_{\mathcal{A}} &= \min\left(\mathsf{Sim}^{l}_{\mathcal{A}}, \mathsf{Sim}^{l'}_{\mathcal{A}}\right), \\ \mathsf{Sim}^{(l,l')}_{\mathcal{O}} &= \min\left(\mathsf{Sim}^{l}_{\mathcal{O}}, \mathsf{Sim}^{l'}_{\mathcal{O}}\right). \end{aligned}$$

From these definitions we can now define a metric between pairs of attribute-predicate instances as

$$\mathsf{Sim}^{A}_{\mathcal{P}}(\,(l,o,v),\,(l',o',v')) = \mathsf{Sim}^{L}_{\mathcal{P}}(l,l') \circledast \mathsf{Sim}^{(l,l')}_{\mathcal{O}}(o,o') \circledast \mathsf{Sim}^{(l,l')}_{\mathcal{A}}(v,v'),$$

The definition of a similarity function between object predicate-instances (l, o_1, o_2) and (l', o'_1, o'_2) on the basis of

1. a $\circledast$-similarity function $\mathsf{Sim}^{L}_{\mathcal{P}}$ defined between pairs of object-predicates in Pred

2. a $\circledast$- similarity function $\mathsf{Sim}^{l}_{\mathcal{O}}$ defined, for every object-predicate l in Pred, between pairs of objects in Obj (i.e., a way to compare possible replacements of the second component of the tuple)

3. a $\circledast$- similarity function $\widehat{\mathsf{Sim}}^{\,l}_{\mathcal{O}}$ defined, for every object-predicate l in Pred, between pairs of objects in Obj (i.e., a way to compare possible replacements of the third component of the tuple)

is given by

$$\mathsf{Sim}^{O}_{\mathcal{P}}(\,(l,o_1,o_2),\,(l',o_1',o_2'))=\mathsf{Sim}^{L}_{\mathcal{P}}(l,l')\circledast\mathsf{Sim}^{(l,l')}_{\mathcal{O}}(o_1,o_1')\circledast\widehat{\mathsf{Sim}}^{(l,l')}_{\mathcal{O}}(o_2,o_2')\,,$$

where

$$\begin{aligned}\mathsf{Sim}^{(l,l')}_{\mathcal{O}} &= \min\left(\mathsf{Sim}^{l}_{\mathcal{O}},\mathsf{Sim}^{l'}_{\mathcal{O}}\right),\\ \widehat{\mathsf{Sim}}^{(l,l')}_{\mathcal{O}} &= \min\left(\widehat{\mathsf{Sim}}^{\,l}_{\mathcal{O}},\widehat{\mathsf{Sim}}^{\,l'}_{\mathcal{O}}\right).\end{aligned}$$

To complete our definition of similarity between predicate instances, we will assume that, having usually rather different meanings, the similarity between any object-predicate instance and any attribute-predicate instance is zero.

A.2.4 Database Editing

We are now in a condition to propose a database-editing methodology to compute the degree of matching between a database and an instantiation of a pattern. Each basic database editing operation, which may include

1. Deletion of binary-predicate instances
2. Addition of binary-predicate instances
3. Modification of binary-predicate instances

transforms a database $\mathcal{D}$ into a modified database $\mathcal{D}'$. If, as discussed earlier, databases are thought of as sets of predicate instances, it is reasonable to measure the degree of adequacy of any transformation employing a measure, based on the underlying metric structures, that gauges the extent to which the knowledge expressed by $\mathcal{D}'$ is consistent with that expressed by $\mathcal{D}$. Such a measure is provided by the degree of implication **I**:

Definition 3: The degree of admissibility of a basic transformation changing a database $\mathcal{D}$ into a database $\mathcal{D}'$ is the degree of inclusion of $\mathcal{D}'$ in $\mathcal{D}$, that is, $\mathbf{I}(\mathcal{D}|\mathcal{D}')$.

We will consider sequences of transformations of the triples in a database that progressively transform the database into a modified, edited, database that matches the pattern. The degree of admissibility of a sequence of basic database editing transformations

$$T=(E_1,E_2,\ldots,E_n)\,,$$

is defined as the composition of the degrees of admissibilities of the component editing transformations, that is,

$$\mathsf{Ad}(T) = \mathsf{Ad}(E_1) \circledast \mathsf{Ad}(E_2) \circledast \ldots \circledast \mathsf{Ad}(E_n)\,.$$

The validity of this aggregation is assured by the transitivity of the degee of inclusion **I** [33].

Degree of matching Several transformations, or sequences of database editing operations, may result in the transformation of a database $\mathcal{D}$ into the same transformed database $\mathcal{D}'$. We may think of each such sequence as a path in a database space between the original and the transformed database. Each path accomplishing the same transformation has an associated admissibility measure that is a function of the admissibility of individual edits. From this perspective, it makes sense to measure the admissibility of a transformation in terms of the path having maximum admissibility.

Definition 4: The **degree of matching** between two databases $\mathcal{D}$ and $\mathcal{D}'$ is the admissibility of the sequence of transformations T mapping $\mathcal{D}$ into $\mathcal{D}'$ having maximum admissibility.

It is important to note that, unlike classical similarity and distance metrics, the degree of matching function defined above will not be, in general, a symmetric function of its two arguments. The major reason for this lack of symmetry lies on the different cost associated with editing operations that are the inverse of each other (e.g., the cost of adding a predicate-instance to a database $\mathcal{D}$ is not the same as that of deleting a database from $\mathcal{D}'$).

Admissibility of basic edit operations In our formulation, the value of admissibility measures for basic database-editing operations depends on the nature of predicate-instances being modified. Whenever needed to introduce new triples, however, new unlinked objects will be added to the database at no cost (i.e., addition of new, unrelated, internal object representations does not entail introduction of unavailable knowledge).

Addition of binary-predicate instances The addition of triples of the form

$$(\,\texttt{predicate}, \texttt{object}_1, \texttt{object}_2\,)\,,$$

results in the replacement of a database $\mathcal{D}$ by the modified database $\mathcal{D}' = \mathcal{D} \cup \{t\}$. As we have already seen, the degree of admissibility of this editing transformation E is given by

$$\mathsf{Ad}(E) = \mathbf{I}(\mathcal{D}|\mathcal{D} \cup \{t\}) = \max_{t' \in \mathcal{D}} \mathsf{Sim}^{O}_{\mathcal{P}}(t, t') .$$

A similar argument leads to the definition of the admissibility $\mathsf{Ad}^{2}_{\mathcal{P}}(T)$ of the transformation that adds an attribute-predicate instance t to the database $\mathcal{D}$ as

$$\mathsf{Ad}(E) = \mathbf{I}(\mathcal{D}|\mathcal{D} \cup \{t\}) = \max_{t' \in \mathcal{D}} \mathsf{Sim}^{A}_{\mathcal{P}}(t, t') .$$

Deletion of binary-predicate instances The deletion of a predicate instance from a database $\mathcal{D}$ results in a database $\mathcal{D}'$ that is a subset of the transformed database.

These transformations are fully admissible, that is,

$$\mathsf{Ad}(E) = \mathbf{I}(\mathcal{D}' \cup \{t\}|\mathcal{D}') = 1 ,$$

since there should not be any cost associated with disregarding information that facilitates the matching between database and pattern. On the other hand, the inverse operation—adding a predicate instance to the database—entails, as we have seen above, the introduction of information that is not supported by prior evidence and its admissibility is measured by the extent to which the assumed information resembles that in the database.

Replacement of binary-predicate instances It is straightforward to prove that the admissibility of a basic editing transformation E of replacing an object-predicate instance t by another object-predicate instance t' is given by the expression

$$\mathsf{Ad}(E) = \max_{''\in \mathcal{D}} \mathsf{Sim}^{O}_{\mathcal{P}}(t', '') .$$

This result, which is consistent with our previous estimates of the admissibility of addition and deletion of triples, shows that the cost associated with the replacement of a triple t by a triple t' is equivalent to the cost of adding t' composed with the cost of deleting t. Since, as we have seen, there is no cost associated with deletions, the cost of replacement is, therefore, simply that of adding the new triple.

A similar result holds for replacement of attribute-predicate instances, that is,

$$\mathsf{Ad}(E) = \max_{t'' \in \mathcal{D}} \mathsf{Sim}^{A}_{\mathcal{P}}(t', t'') .$$

A.3 Imprecision, Uncertainty, Vagueness

Our discussion has focused, so far, on the nature of databases that are conventional in the sense that, whenever the value of the attribute of an object is specified, such a value is a unique element of the range of possible values of the attribute. Similarly, if two objects are related, or linked, there is no ambiguity as to their identity.[21] We start our exposition by briefly sketching the sense in which terms such as "imprecise," "uncertain," or "vague" are used in our exposition.

Although a convention is yet to be reached among practitioners as to the proper usage of the terms *imprecision* and *uncertainty*, throughout this discussion we will describe imprecision as the inconvenient characteristic of information that does not allow identification of the value of an attribute or does not permit unique identification of objects having a relationship. Rather, it will be assumed that imprecise information permits us to identify a set of possible attribute values or a set of objects where the actual attribute value or the related object lies, respectively.

The following assertions exemplify imprecise knowledge:

> "The `age` of person `P-3` is at least `20 years`,"
> "The `father` of person `P-99` is either person `P-100`, or person `P-102`."

The key feature of imprecision is the inability to specify actual values of attributes or to permit unique identification of objects, allowing, rather, identification of a subset of the range of a particular attribute or relation.

We also discuss possible imprecision about the nature of the links that define the property being described or the nature of the relation between two objects. It may only be known, for example, that

> "Person `P-46` is a `Relative` of Person `P-3` ,"

while better knowledge may reveal that `P-46` is the `Father` of `P-3` , that is, a more precise characterization of the link between persons.

The term *uncertainty* is usually employed in the literature to describe probabilistic knowledge about the value of an attribute or about the identity of linked objects as exemplified by

> "The probability that the `total-rainfall` will be `50in` is `50%`."
> "The probability that the `father` of `P-90` is `P-3` is 90%."

[21] As we have noted, however, our previous treatment is general enough to be applicable to incomplete databases, where values of certain properties may not be specified, or where, similarly, certain relation instances may be missing.

essentially defining elements of a probability distribution in the range of a property or relation (i.e., the conditional probability that a property has a value, or the conditional probability that an object is related to another object, *given available evidence*). The basic difference between imprecise and uncertain knowledge is that the former simply specifies a subset of possible values or related objects[22] while the latter fully specifies a probability distribution over the range of the property or relation [50].

In this report, primarily for the sake of clarity, we confine our attention to pattern-matching problems involving imprecise information. While the extension to uncertain information is relatively straightforward, it involves the introduction of functions defining distances and similarity between probability distributions, resulting in the consideration of issues that are not central to the problem being addressed. For the same reason, we also avoid discussion of generalized schemes, such as the *Dempster-Shafer calculus of evidence* [40] (which relies on a combination of imprecise and uncertain representations of partial knowledge) nor do we deal with extensions to the *fuzzy* domain (which are concerned with the representation of vague information).

A.3.1 Imprecise Objects, Values, and Relations

The introduction of imprecision in data models adds considerable complexity to the nature of the problems that must be considered in the context of a pattern-matching application. The lack of accepted methodologies for the representation of imprecise knowledge is a major obstacle to straightforward theoretical extensions. Reliance on conventional mechanisms for identification of objects on the basis of the values of their attributes, such as *keys*, can no longer be employed, since imprecision on the values of properties or in the identity of related objects introduces ambiguities on the nature of the objects being modeled. Furthermore, as we will examine in some detail, the metric structures required for determination of the degree of admissibility of a transformation are no longer symmetric—properly reflecting the different costs associated with a transformation and its inverse (e.g., replacing `Animal` by `Cow` introduces new knowledge while the converse transformation does not).

[22] This specification may be thought of as stating that the probability that the value of a related attribute or object lies on some subset is 1, i.e., a partial specification of a probability distribution.

Imprecise databases Our previous database model will now be extended to permit the representation of imprecise knowledge about the nature of links, related objects, and attribute values.

Definition 5: A data model is a 4-tuple

$$\mathcal{D} = (\,\mathsf{Obj}, \mathsf{Vals}, \mathsf{Pred}, \mathsf{Data}\,),$$

where

(i) *Objects*: Obj is a nonempty set, called the set of *objects*.

While we will not introduce any substantive changes to our previous characterization of the set Obj, we will assume, however, that it is possible to describe a finite set of objects simply by defining its members, i.e.,

$$\texttt{Object-Set} = \{\,\texttt{Object}_1, \ldots, \texttt{Object}_n\,\}.$$

This capability permits the representation of imprecise knowledge about the nature of objects related to some object instance, such as

```
(Father, P12, {P23, P34}),
```

representing knowledge that the `Father` of person `P12` is either `P23` or `P34`.

(ii) *Values*: Values are nonempty sets in a collection of basic primitive types.

Once again, we will assume that it is possible to represent finite sets of property values, i.e.,

$$\texttt{Value-Set} = \{\,\texttt{Value}_1, \ldots, \texttt{Value}_n\,\}.$$

This capability permits the representation of imprecise knowledge about the values of an attribute of an object instance, such as

```
(hasCityAddress, P12, { "Menlo Park", "Palo Alto"
}),
```

representing knowledge that person `P12` lives either in Palo Alto or in Menlo Park.

In the case of sets in the collection Vals, however, we will assume that there usually exist domain-dependent, taxonomical structures, defined in those

sets that permit characterization, for example, that the Nationality of a Person is European (i.e., as opposed to specifying a particular country) or that the Age of a Person is between 20 and 30 years.

We will denote by $\mathsf{Leaf}_{\mathcal{V}}$ the collection of all *leaf nodes* of all taxonomical structures defined in sets belonging to Vals. This set will contain, for example, all atomic strings, integer numbers, and members of sets such as European that do not have any successors (e.g., Italian).

We will further assume that any taxonomical entry in Vals is equivalent, in a set-theoretical sense, to the set of its descendant leaf nodes. For example, the value European is equivalent to the set of all specific European nationalities, i.e., {Spanish, Italian, French, ...}

(iii) *Predicates*: Pred is a nonempty set, called the set of *predicates*, or the set of *links*. Again, we assume that there exists a similarity function $\mathsf{S}_{\mathcal{L}}$ defined between pairs of predicates.

We will now assume, however, that there exists a hierarchical order between links defining, as was the case with values, a generalized taxonomical structure between links. While in our previous treatment, all predicates were distinct members of the set Pred, we now allow the possibility that predicates may be related by the notion of logical implication. For example

$$\mathsf{Son} \Rightarrow \mathsf{Child} \Rightarrow \mathtt{Relative}\,,$$

exemplifies predicates that define different levels of imprecision, or *granularity*, of relations between persons.

As was also the case with values, we will denote by $\mathsf{Leaf}_{\mathcal{L}}$ the set of all leaf nodes of Pred, i.e., the set of all *atomic*, or *ground*, predicates. Similarly, we will equate each predicate Predicate in Pred with the set of its descendant leaf nodes, that is, the set of all atomic predicates predicate that imply Predicate.

(iv) *Predicate Instances*: We will need to consider again two classes of predicates, called *binary object predicates* and *binary attributes predicates*, respectively.

Here, however, we will introduce substantive generalizations by permitting the expression of ambiguity about the nature of the links, about the objects that are related to a uniquely identified object-instance, or about the value of the attribute of an object-instance.

Instances of generalized object predicates will now be expressed by triples such as

```
(Predicate, object-instance, Object-Set)
```

which state that a uniquely specified object instance is related, by a possibly ambiguous object predicate, to some member of `Object-set`. It is important to note that it is not necessary to permit ambiguity in the specification of the second component since, as is well known, a many-to-many relation may be specified as a collection of one-to-many relations.

Instances of generalized attribute predicates are now expressed by triples such as

```
(Predicate, object-instance, Value-Set)
```

which specify that the value of a possibly ambiguous attribute of the specified object instance lies in the set `Value-Set`.

We will assume again that certain functions, defined in the sets $\mathcal{P}_O$ and $\mathcal{P}_A$ of predicate instances, provide the rational bases to measure the admissibility of database editing operations. As we will see below, however, these functions are no longer similarity measures but, rather, nonsymmetric *generalized order* functions.

(v) *Data*: The data Data is a tuple $(\mathsf{Data}_{\mathcal{O}}, \mathsf{Data}_{\mathcal{A}})$, where $\mathsf{Data}_{\mathcal{O}}$ is a set of generalized object-predicate instances and $\mathsf{Data}_{\mathcal{A}}$ is a *nonempty* generalized attribute-predicate instances. The only difference, albeit significant, with our previous formulation lies in the ability to accommodate imprecision, either as ambiguous related objects or as ambiguous property values, provided by generalized object-predicate instances and attribute-predicate instances, respectively. Furthermore, the first triple component, i.e., `Predicate`, may itself be ambiguous.[23]

Generalized metrics Consideration of imprecision introduces significant differences in the numeric structures employed to measure semantic differences between databases. Once again, it will be necessary to define, on a rational semantic

[23] We have chosen to capitalize the first letter of the term `Predicate` to indicate that now links may be ambiguous.

basis, the admissibility functions $\mathsf{Ad}^O_{\mathcal{P}}$ and $\mathsf{Ad}^A_{\mathcal{P}}$ that measure the adequacy of basic database editing transformations. This definition will also be based on metric structures defined now in the sets $\hat{\mathcal{P}}_O$ and $\hat{\mathcal{P}}_A$ of generalized object and attribute predicate instances.

To facilitate understanding of the basic issues, we discuss first extensions to the editing of generalized attribute predicates. We will assume now that there exists a similarity function $\mathsf{Sim}^L_{\mathcal{P}}$ defined between pairs of attribute-predicates in $\mathsf{Leaf}_{\mathcal{L}}$. This similarity function generalizes our previous metric that, in the case of precise databases, was defined between any pair of predicates in Pred.

We will assume now that for every leaf attribute-predicate l in $\mathsf{Leaf}_{\mathcal{L}}$, there exists

1. A similarity function $\mathsf{Sim}^l_{\mathcal{O}}$ defined between pairs of objects in Obj (i.e., a way to compare possible replacements of the second component of the tuple)

2. A similarity function $\mathsf{Sim}^l_{\mathcal{A}}$ defined between pairs of leaf values in the set $\mathsf{Leaf}_{\mathcal{V}}$ (i.e., a way to compare possible replacements of the third component of the tuple)

As we have previously noted, these metric structures provide the bases to define a similarity function between triples representing attribute-predicate instances as

$$\mathsf{Sim}^A_{\mathcal{P}}(\,(l,o,v),\,(l',o',v')) = \mathsf{Sim}^L_{\mathcal{P}}(l,l') \circledast \mathsf{Sim}^{(l,l')}_{\mathcal{O}}(o,o') \circledast \mathsf{Sim}^{(l,l')}_{\mathcal{A}}(v,v')\,.$$

This definition, however, must now be generalized to provide a basis to determine the admissibility of replacing a generalized attribute-predicate instance (L,o,V) by another (L',o',V'), where V and V' are subsets of a values and where L and L' are generalized predicates.

Clearly, the similarity function $\mathsf{Sim}^l_{\mathcal{A}}$, defined between pairs of leaf, or atomic, values v and v' must be extended to a metric over pairs of value sets V and V'. Although it is possible to extend similarity structures defined over a set to metrics defined over its power set (i.e., the set of all its subsets) by means of the well-known Hausdorff distance [33], with the formula

$$\mathsf{Sim}_H(V,V') = \min\,[\,\mathbf{I}(V|V'),\,\mathbf{I}(V'|V)\,]\,,$$

where $\mathbf{I}$ is the degree of inclusion function associated with the similarity function Sim, it is important to remember that our objective is to characterize the extent

to which it is admissible to replace, in a database $\mathcal{D}$, a triple (L, o, V) by another triple (L', o', V').

As we will see below, this is a non-symmetric transformation—since, generally, the admissibility of replacing (L, o, V) by (L', o', V') is not the same as that of replacing (L', o', V') by (L, o, V). As it turns out, we will only need to resort, again, to degree-of-inclusion measures $\mathbf{I}$ to derive the required admissibility costs.

We focus now, for the sake of explanation, on the simpler problem of replacing *only* an imprecise value V by another imprecise value V' in a triple of the form (l, o, V), where l is a precise predicate and o is a specific object (i.e., determining the admissibility of replacing the triple (l, o, V) by the triple (l, o, V')). To estimate the extent to which it is acceptable to replace the third component V by V', we need to remember that the value-set V represents the best evidence as to the nature of the actual value v of the attribute l of an object O. Assuming now that the "true value" of that attribute is the element (i.e. the leaf value) v of V, the extent to which V' is a proper description of the nature of v is

$$\max_{v' \in V'} \mathsf{Sim}^l_{\mathcal{A}}(v, v') \,.$$

In other words, if v were to lie actually in V', then it should be acceptable—from a strictly truth-oriented perspective,[24] despite a possible loss of information—to say that the attribute l of object O lies in V'. On the other hand, when every v' in V' is very different from v, then all the values $\mathsf{Sim}^l_{\mathcal{A}}(v, v')$ are small and, accordingly, the admissibility of representing the location of v by V' is low.

On the basis of these considerations, since, in fact, we only know that the true value v lies in V, it is clear that the minimum possible degree of admissibility of replacing V by V', is given by

$$\min_{v \in V} \max_{v' \in V'} \mathsf{Sim}^l_{\mathcal{A}}(v, v') = \mathbf{I}^l_A(V'|V) \,.$$

Note, in particular, that this expression indicates that it is admissible to replace V by another set V' that includes it although the opposite is not true. This lack of symmetry is consistent with the meaning of the transformation (i.e., there is no loss of "truth" in saying that a city lies in the United States when it is known that it is in California but the converse statement is obviously false).

[24] It is important to remember that our editing transformations are not intended to eliminate ambiguity but, rather, to produce a transformed database that matches the conditions imposed by the pattern.

We can now, generalize our characterization of the degree of similarity between instances of attribute predicates by the function

$$\mathbf{I}^{A}_{\mathcal{P}}(\,(L,o,V)\,|\,(L',o',V'))=\min_{l\Rightarrow L}\,\max_{l'\Rightarrow L'}\,\left[\,\mathsf{Sim}^{L}_{\mathcal{P}}(l,l')\circledast\mathsf{Sim}^{(l,l')}_{\mathcal{O}}(o,o')\circledast\mathbf{I}^{(l,l')}_{A}(V'|V)\,\right]\,,$$

where $\mathbf{I}^{(l,l')}_{A}$ is the degree of inclusion function associated with the similarity function $\mathsf{Sim}^{(l,l')}_{\mathcal{A}}$. The function $\mathbf{I}^{A}_{\mathcal{P}}$, not being symmetric, is not a similarity measure but actually a generalized *order* function, that is, a reflexive and $\circledast$-transitive function.

Turning our attention now to generalization, we will again assume that there exists a similarity function $\mathsf{Sim}^{L}_{\mathcal{P}}$ defined between pairs of object-predicates in $\mathsf{Leaf}_{\mathcal{L}}$ and that, for every leaf object-predicate l in $\mathsf{Leaf}_{\mathcal{L}}$, there exists

1. A similarity function $\mathsf{Sim}^{l}_{\mathcal{O}}$ defined between pairs of objects in Obj (i.e., a way to compare possible replacements of the second component of the tuple)

2. A similarity function $\widehat{\mathsf{Sim}}^{\,l}_{\mathcal{O}}$ defined between pairs of objects in Obj (i.e., a way to compare possible replacements of the third component of the tuple)

By an argument identical to that employed above in the case of generalized attribute-predicate instances we can now define the required order function as

$$\mathbf{I}^{O}_{\mathcal{P}}\,(\,(L,o_1,O_2)\,|\,(L',o'_1,O'_2))=\min_{l\Rightarrow L}\,\max_{l'\Rightarrow L'}\,\left[\mathsf{Sim}^{L}_{\mathcal{P}}(l,l')\circledast\mathsf{Sim}^{(l,l')}_{\mathcal{O}}(o_1,o'_1)\circledast\widehat{\mathbf{I}}^{(l,l')}_{O}(O'_2|O_2)\,,\right]\,,$$

where $\widehat{\mathbf{I}}^{(l,l')}_{O}$ is the degree of inclusion function associated with the object-similarity measure $\widehat{\mathsf{Sim}}^{(l,l')}_{\mathcal{O}}$.

To complete our definition of similarity between predicate instances, we will assume, as before, that, having rather different meanings, the similarity between any generalized object-predicate instance and any generalized attribute-predicate instance is zero.

Admissibility of basic editing operations We can now state formulas for the admissibility of basic editing operations. The arguments leading to these expressions are identical to that given earlier for precise predicate instances and will not be repeated here.

Addition of binary-predicate instances The admissibility of the basic editing transformation E consisting of the addition of a generalized object-predicate triple T of the form

$$T = (\,\texttt{Predicate}, \texttt{object}_1, \texttt{Object-set}_2\,)\,,$$

resulting in the replacement of a database $\mathcal{D}$ by the modified database $\mathcal{D}' = \mathcal{D} \cup \{T\}$ is given by

$$\mathsf{Ad}(E) = \max_{T' \in \mathcal{D}} \mathbf{I}^{O}_{\mathcal{P}}(T|T')\,.$$

Correspondingly, the admissibility of the basic editing transformation E consisting of the addition of a generalized attribute-predicate triple T of the form

$$T = (\,\texttt{Predicate}, \texttt{object}, \texttt{Value-set}\,)\,,$$

resulting in the replacement of a database $\mathcal{D}$ by the modified database $\mathcal{D}' = \mathcal{D} \cup \{T\}$ is given by

$$\mathsf{Ad}(E) = \max_{T' \in \mathcal{D}} \mathbf{I}^{A}_{\mathcal{P}}(T|T')$$

Deletion of binary-predicate instances The deletion of a predicate instance from a database $\mathcal{D}$ results in a database $\mathcal{D}'$ that is a subset of the transformed database. These transformations are, as was previously the case, fully admissible, that is,

$$\mathsf{Ad}(E) = 1\,,$$

since there should not be any cost associated with disregarding information that facilitates the matching between database and pattern.

Replacement of binary-predicate instances On the basis of arguments that are identical to those given when discussing the replacement of an object-predicate instance t by another object-predicate instance t', the admissibility of the database transformation E replacing a generalized object-predicate instance T by another object-predicate instance T' is given by

$$\mathsf{Ad}(E) = \max_{T'' \in \mathcal{D}} \mathbf{I}^{O}_{\mathcal{P}}(T''|T')\,.$$

Similarly, when replacing a generalized attribute-predicate instance T by a generalized attribute-predicate instance T', the admissibility of the corresponding database transformation E is given by

$$\mathsf{Ad}(E) = \max_{T'' \in \mathcal{D}} \mathbf{I}^{A}_{\mathcal{P}}(T''|T')\,.$$

References

[1] D. Allsopp, P. Beautement, J. Carson, and M. Kirton. Toward semantic interoperability in agent-based coalition command systems. In *Proceedings of the International Semantic Web Working Symposium*, 2002.

[2] Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, and Katia Sycara. DAML-S: Web service description for the semantic web. In *Proceedings of the First International Semantic Web Conference (ISWC)*, Sardinia, Italy, 2002.

[3] Roland Backhouse and Paul Hoogendijk. Elements of a relational theory of datatypes. In Bernhard Moeller, Helmut Partsch, and Steve Schuman, editors, *Formal Program Development*, volume 755, pages 7–42. Springer-Verlag, New York, N.Y., 1993.

[4] Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19/20:73–148, 1994.

[5] P.M. Berry and B. Drabble. SWIM: An AI-based system for workflow enabled reactive control. In *Proceedings of the IJCAI Workshop on Workflow and Process Management*, Stockholm, Sweden, 1999.

[6] James C. Bezdek and Sankar K. Pal, editors. *Fuzzy Models for Pattern Recognition: Methods that Search for Structures in Data*. IEEE Press, 1992.

[7] Alexander Budanitsky and Graeme Hirst. Semantic distance in WordNet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and Other Lexical Resources, Second Meeting of the North American Chapter of the Association for Computational Linguistics*, Pittsburgh, PA, 2001.

[8] H. Bunke. On a relation between graph edit distance and maximum common subgraph. *Pattern Recognition Letters*, 18:689–694, 1997.

[9] H. Bunke and K. Shearer. A graph distance metric based on the maximal common subgraph. *Pattern Recognition Letters*, 19:255–259, 1998.

[10] V. K. Chaudhri, A. Farquhar, R. Fikes, P. D. Karp, and J. P. Rice. OKBC: A programmatic foundation for knowledge base interoperability. In *Proceedings of the AAAI-98*, Madison, WI, 1998.

[11] P.P.-S. Chen. The entity-relationship model – toward a unified view of data. *IACM Transactions on Database Systems*, pages 9–36, 1976.

[12] Adam Cheyer and David Martin. The Open Agent Architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 4(1):143–148, March 2001.

[13] A. Colmerauer. An introduction to Prolog-III. *Communications of the ACM*, 330:69–90, 1990.

[14] Cycorp. Opencyc documentation.

[15] Cycorp. Ontological engineer's handbook, 2003. http://www.opencyc.org.

[16] C. J. Date. *Introduction to Database Systems*. Addison-Wesley, sixth edition, 1995.

[17] Daniel J. Dougherty and Claudio Gutierrez. Normal forms and reduction for theories of binary relations. In *RTA*, pages 95–109, 2000.

[18] Didier Dubois and Henri Prade. An introduction to possibilistic and fuzzy logics. In Didier Dubois, Henri Prade, and P. Smets, editors, *Non-Standard Logics for Automated Reasoning*. Academic Press, 1988.

[19] I. A. Harrison. ControlML schema design, 2002. http://www.ai.sri.com/l̃aw/schemas/2002/07/controlML.htm.

[20] I. A. Harrison. HypothesisML schema design, 2002. http://www.ai.sri.com/l̃aw/schemas/2002/07/hypothesisML.htm.

[21] I. A. Harrison. PatternML schema design, 2002. http://www.ai.sri.com/l̃aw/schemas/2002/07/patternML.htm.

[22] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, SSC-4(2):100–107, 1968.

[23] J. Hendler and D. L. McGuinness. The DARPA agent markup language. *IEEE Intelligent Systems*, pages 67–73, November 2000.

[24] J. Kubica, A. Moore, J. Schneider, and Y. Yang. Stochastic link and group detection. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, 2002.

[25] J. D. Lowrance, I. W. Harrison, and A. C. Rodriguez. Capturing analytic thought. In *Proceedings of the First International Conference on Knowledge Capture*, pages 84–91, October 2001.

[26] John McCarthy. LISP 1 programmer's manual. Technical report, Computation Center and Research Laboratory of Electronics, MIT, Cambridge, Mass., 1960.

[27] Karen L. Myers. *The ACT Editor User's Guide*. Artificial Intelligence Center, SRI International, Menlo Park, CA, 1993.

[28] Karen L. Myers. *User's Guide for the Procedural Reasoning System*. Artificial Intelligence Center, SRI International, Menlo Park, CA, 1993.

[29] K.L. Myers and David.N. Morley. *Adjustable Autonomy*, chapter Policy-based Agent Directability. MIT Press, 2002.

[30] S. M. Paley, J. D. Lowrance, and P. D. Karp. A generic knowledge-base browser and editor. In *Proceedings of the 1997 National Conference on Artificial Intelligence*, 1997.

[31] Vaughan R. Pratt. Origins of the calculus of binary relations. In *Logic in Computer Science*, pages 248–254, 1992.

[32] H. Raiffa. *Decision Analysis*. Addison-Wesley, 1968.

[33] E. H. Ruspini. On the semantics of fuzzy logic. *Int. J. of Approximate Reasoning*, 5:45–88, 1991.

[34] E. H. Ruspini. Truth as utility: A conceptual synthesis. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 316–322, Los Angeles, CA, 1991.

[35] E. H. Ruspini, P. P. Bonissone, and W. Pedrycz, editors. *The Handbook of Fuzzy Computation*. Institute of Physics, 1998.

[36] E H. Ruspini and F. Esteva. Interpretations of fuzzy sets. In E. H. Ruspini, P. P. Bonissone, and W. Pedrycz, editors, *The Handbook of Fuzzy Computation*. Institute of Physics, 1998.

[37] Enrique H. Ruspini. On truth and utility. In Rudolf Kruse and Pierre Siegel, editors, *Symbolic and Quantitative Approaches for Uncertainty: Proceedings of the European Conference ECSQAU, Marseille, France, October 1991*, pages 297–304. Springer-Verlag, Berlin, 1991.

[38] Enrique H. Ruspini and John Lowrance. Semantic indexing and relevance measures in SEAS. Working Paper (DARPA GENOA Project, 2002.

[39] A. Saffiotti, K. Konolige, and E.H. Ruspini. A multivalued logic approach to integrating planning and control. *Artificial Intelligence*, 76:481–526, 1995.

[40] Glenn Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.

[41] L. Shapiro and R. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:504–519, 1981.

[42] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1983.

[43] J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley, 1984.

[44] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Eighteenth Conference on Uncertainty in Artificial Intelligence*, Edmonton, Canada, 2002.

[45] J. Thomere, I. Harrison, J. Lowrance, A. Rodriguez, E. Ruspini, and M. Wolverton. Helping intelligence analysts detect threats in overflowing, changing, and incomplete information. In *IEEE International Conference on Computational Intelligence for Homeland Security and Personal Safety (CIHSPS2004)*, 2004.

[46] J. Thomere, A. Rodriguez, V. Chaudhri, S. Mishra, M. Eriksen, P. Clark, K. Barker, and B. Porter. A web-based ontology browsing and editing system. In *Conference on Innovative Applications of Artificial Intelligence*. AAAI, Jul 2002.

[47] L. Valverde. On the structure of F-indistinguishability operators. *Fuzzy Sets and Systems*, 17(3):313–328, 1985.

[48] M. Wolverton. *Retrieving Semantically Distant Analogies*. PhD thesis, Stanford University, 1994.

[49] M. Wolverton, P. Berry, I. Harrison, J. Lowrance, D. Morley, A. Rodriguez, E. Ruspini, and J. Thomere. LAW: A workbench for approximate pattern matching in relational data. In *The Fifteenth Innovative Applications of Artificial Intelligence Conference (IAAI-03)*, 2003.

[50] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.