

AD-A286 363



Performance Measurement and Analysis
of Certain Search Algorithms

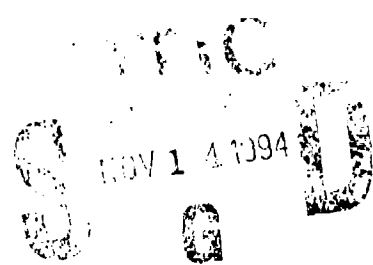
94-34921



3118

John Gaschnig

May 1979



Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pennsylvania 15213

Submitted to Carnegie-Mellon University in
partial fulfillment of the requirements for the
degree of Doctor of Philosophy.

Copyright -C- 1979 John G. Gaschnig

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

ABSTRACT

This thesis applies the methodology of analysis of algorithms to study certain combinatorial problems and search algorithms originating predominantly in the AI literature, and extends that methodology to include experiments in a complementary role.

Chapters 2 and 3 combine experimental and analytic techniques respectively to measure and to predict the performance of the A* best-first search algorithm, which solves path-finding problems defined in terms of finite strongly connected graphs. In this domain, we make numerous experimental performance measurements varying the heuristic function, the size of the problem, a weighting coefficient, and the performance measure; we derive general formulas in a simpler worst case analysis model that purport to predict the experimental observations when evaluated at particular argument values that correspond to the experimental parameter settings; and we test the analytic predictions against the experimental observations. The A* experiments use as case study a randomly generated set of instances of the "Eight" puzzle of varying size (depth of goal). The analysis in Chapter 3 extends the worst case tree search model of Pohl and others to arbitrary heuristic functions, resulting in cost formulas whose arguments include functions.

Chapter 4 reports experimental results for a second problem domain, that of a class of satisficing assignment problems. Here we measure and compare under varying conditions the performances of four functionally equivalent algorithms -- the so-called backtrack algorithm, a version of the so-called "network consistency" or constraint satisfaction algorithm of Waltz, and two new algorithms BACKMARK and BACKJUMP. The experiments span four case studies: two sets of N-queens problems and two sets of randomly generated problems whose characteristics are specified by the values of certain parameters. Note that we are not interested primarily in the 8-puzzle or in the N-queens problems per se, but rather as relatively simple yet non-trivial case studies in which to explore general issues with rigor, principally the issue of predicting algorithm performance.

The results take a number of forms: they variously confirm, disagree with or qualify hypotheses about algorithm performance found in the literature; tens of thousands of algorithm executions reveal new phenomena about algorithm performance; new algorithms are devised based on insights obtained from performance evaluation.

x

A-1

ACKNOWLEDGEMENTS

I will remain long and deeply indebted to Herb Simon, H. T. Kung, Al Newell, Jay Kadane, and Michael Shamos, who constituted my thesis committee, especially to Herb, its chairman and my advisor of long standing. Their high standards, their time, their patience, and their interest helped make this thesis more than it otherwise might have been. My officemates Don Kosy and Roy Levin were ever-willing and much-appreciated sounding boards. I am also grateful to many others who listened and responded at various times. Many thanks are due to those who developed and maintain the excellent hardware and software facilities of the department. The friendliness, energy, and spirit of community that pervades the Computer Science Department provided an environment both conducive to research and fun to be a part of. My friends at Shady, Wilkins, Northumberland, Reynolds, and Northumberland enriched my life and made Pittsburgh a wonderful home; no few words can convey my appreciation of what we shared together. To all encouraging friends in Pittsburgh and California, thanks -- it helped.

Title _____	
Author _____	
Date _____	
By _____	
Distribution _____	
Availability Codes	
Dist	Available For Special
A-1	_____

TABLE OF CONTENTS

1 Introduction and Overview

1.1 Predictive, Experimentally Testable Theories in Artificial Intelligence	1
1.2 Objectives, Methodology, and Scope	
1.2.1 Objectives	5
1.2.2 Methodology	7
1.2.3 Scope	7
1.3 Computational Models: Defining Problems, Algorithms, Heuristics	
1.3.1 Problem Specification Parameters and Control Policy Parameters	8
1.3.2 Analytic Predictions vs. Experimental Observations	12
1.3.3 Abstractions: Monotonicity Theorems on a Lattice of Algorithms	13
1.4 Examples of General Questions Modeled in a Restricted Context	
1.4.1 How Much Does "Parameter Tuning" Change Performance?	14
1.4.2 How Much "Knowledge" Buys How Much Performance?	15
1.4.3 How to Measure "Structure" in a Problem?	16
1.5 Tradeoffs: Why These Experiments, Why This Analysis?	16
1.6 A Note on Reading This Dissertation	18

2 Experimental Performance Measurement of A*: A Case Study with the "Eight" Puzzle

2.0 Summary of Chapter	20
2.1 Introduction	22
2.2 Cost and Solution Quality for 8-Puzzle Heuristics	24
2.3 Parameter Tuning: Effects of Changing Term Weighting	35
2.4 Cost vs. Error in Heuristic Estimates of Distance to the Goal	39
2.5 "Internal" Measures of Search Behavior	43
2.6 Predictions About Performance of a Complex Best-first Search System	45
2.7 Procedure for Generating Random Problem Instances	47
2.8 Statistical Issues	51
2.9 Conclusions and Future Experiments	55
Figures for Chapter 2	58

3 Worst Case Cost of A* as a Function of Error in Heuristic Distance Estimates

3.0 Summary of Chapter	73
3.1 A Distance-Estimating, Bounded-Estimate Tree Search Model (DEBET)	
3.1.1 Introduction	75
3.1.2 First Definitions	76
3.1.3 A General Case Theorem: Which Nodes are Expanded?	79
3.1.4 The $\langle KMIN, KMAX \rangle$ Model of Heuristic Functions: Definitions	81
3.1.5 Bounds on Heuristic Distance Estimates Imply Bounds on Lengths of "Garden Paths": The $YMAX(KMIN, KMAX, W, r)$ Function	85
3.1.6 A General Case Theorem: How Many Nodes are Expanded?	85
3.2 Simplifying Assumptions for Analysis	
3.2.1 Definitions and Lemmas	88
3.2.2 A Theorem Simplifying the Computation of $YMAX(KMIN, KMAX, W, r)$	90
3.2.3 Monotonicity Theorems: Comparing Two Heuristic Functions	91
3.2.4 Applications to the Class of "Linearly Bounded" Heuristic Functions	
3.2.4.1 Simple Formulas and Their Geometric Interpretation	92
3.2.4.2 A Scalar Optimization Operation on Linearly-Bounded Heuristics	95
3.3 Cost as a Function of Relative Error in Heuristic Estimates	
3.3.1 Definitions	96
3.3.2 A Theorem Relating "Garden Path" Length to Relative Error	97
3.3.3 A Simple Formula Bounding Cost as a Function of Relative Error	99
3.3.4 Theorems: Cost Grows Monotonically with Relative Error	102
3.3.5 Lattice Formulation	104
3.4 Parameter Tuning: When is Insurance Justified?	
3.4.1 Introduction	105
3.4.2 Theorem: $W = .5$ is Optimal for "IM-Never-Overestimating" Heuristics	106
3.4.3 W -Optimality for "Linearly Bounded" Heuristic Functions	108
3.5 Analytic Predictions vs. Experimental Measurements for 8-Puzzle Heuristics	
3.5.1 Numerical Comparisons	109
3.5.2 Comments	112
3.6 Conclusions and Open Problems	114
3.7 DEBET Results as a Step Toward a Theory About the Relation of "Knowledge" to Performance	117
Figures for Chapter 3	121

4 Experimental Case Studies of Backtrack vs. Waltz-type vs. New Algorithms for Satisficing Assignment Problems

4.0 Summary of Chapter	142
4.1 Backtrack vs. Waltz-type Algorithms: What to Measure and Why	
4.1.1 Definitions, Examples, and Elementary Results	145
4.1.2 "Obvious" vs. Random Orderings of Candidate Values	158
4.1.3 Analysis of Waltz' Experimental Results	161
4.2 New General Algorithms Combining Backtrack and Constraint Satisfaction	
4.2.1 BACKMARK: Backtrack With Fewer Redundant Pair-tests	163
4.2.2 BACKJUMP: Backtrack that Jumps Multiple Levels	168
4.2.3 DEELEV(i): Constraint Satisfaction After Backtracking to Level i	171
4.3 Comparative Performance Measurements for N-Queens SAPs	174
4.4 Experimental Results for Randomly Generated SAPs	
4.4.1 SAP Equivalence Classes Parameterized by Size and by "Degree of Constraint" (L)	176
4.4.2 N-Queens SAPs vs. "Random-N-Queens" SAPs: Comparative Algorithm Performance	178
4.4.3 Cost as a Function of L: A Sharp Peak at $L \sim 0.6$	179
4.5 Other Results	
4.5.1 Experimental Results for Map Coloring	181
4.5.2 Measures of Uniformity of Distribution of Solutions	182
4.5.3 Proof that $T_{\min}(N) = N(N-1)/2$	183
4.5.4 Improvements to Mackworth's Version of Waltz Algorithm	185
4.6 Conclusions and Future Experiments	188
Figures for Chapter 4	190

5 Description of Apparatus for Search Experiments

5.0 Summary of Chapter	212
5.1 Issues: Generality, Efficiency, Data Collection and Analysis, Modifiability, General Human Engineering	213
5.2 ASTAR (A* and Variants)	214
5.3 BKDEE (A Family of Backtrack and Constraint Satisfaction Algorithms)	218
5.4 Issues for Future Apparatus	224

6 Conclusions and Future Work

6.1 Contributions

6.1.1 Previous Conjectures Tested Against Hard Data	226
6.1.2 Practical Applications: New Algorithms and Practical Predictions	229
6.1.3 A "Successive Set Partitioning" Approach to Problem "Structure"	232
6.1.4 Analysis of the DEBET "Arbitrary Heuristic" Model of Worst Case A* Tree Search	233
6.1.5 Experimental Tests of Predictive Power of the DEBET Model of A*	234
6.1.6 Abstractions in Analysis of Algorithms	235
6.1.7 10^4 Experimental Observations For Future Theories to Predict and Explicate	237
6.1.8 Cross-domain Comparisons	237

6.2 Immediate Extensions

6.2.1 Mathematical Analysis of Algorithms for Satisficing Assignment Problems	238
6.2.2 Analogous Experiments With Different Problems	239
6.2.3 "Successive Set Partitioning": More Parameters	241
6.2.4 Further Analysis of the DEBET Model of A*	242
6.2.5 Methodological and Practical Issues for Experiments	243
6.2.6 Other Issues Excluded from the Dissertation	245

6.3 Long Term Objectives

6.3.1 Error Analysis in Cross-model Comparisons	248
6.3.2 Algorithm Behavior: What is Observable? What is Controllable?	248
6.3.3 The 9-puzzle as a Highly Regular Graph	249
6.3.4 Toward Theories About "Problem Structure" and "Heuristic Knowledge"	250
6.3.5 Performance Analysis in AI Research: General Comments	251

References	253

Appendix A. Glossary of Terms and Symbols	261

Appendix B. Direct Extensions of Present Experiments and Analysis 266

Appendix C. Tabulation of Experimental Data Plotted in the Figures 274

Chapter 1

Introduction and Overview

One has the sense that the men who conceived these high buildings [Gothic cathedrals] were intoxicated by their new-found command of the force in the stone. How else could they have proposed to build vaults of 125 feet and 150 feet at a time when they could not calculate any of the stresses?

Jacob Bronowski, *The Ascent of Man*

1.1 Predictive, Experimentally Testable Theories in Artificial Intelligence

This dissertation is based on the premise that in the future more of the subject matter of artificial intelligence (AI) research will be understood mathematically than at present. We present the results of limited steps toward that long term objective, here focusing on the performance of certain search algorithms. In brief, we apply the methodology of analysis of algorithms to study certain relatively simple combinatorial problems and search algorithms originating predominantly in the AI literature, and we extend that methodology to include experiments in a complementary role. Two problem solving domains are considered: path-finding search in graphs and trees using the A* best-first search algorithm, and a class of satisficing assignment problems.

The usefulness of a scientifically sound experimental methodology in AI research has been well established for some time:

"Our research strategy in studying complex systems is to specify them in detail, program them for digital computers, and study their behavior empirically by running them with a number of variations and under a variety of conditions. This appears at present the only adequate means to obtain a thorough understanding of their behavior." [Newell, Shaw & Simon 1963, p. 110]

Since the time of that quotation, many algorithms have been analyzed mathematically, but applying analysis of algorithms techniques to complex AI systems remains difficult. In the case of certain search algorithms, we shall attempt to show that both analysis and experiment are useful.

This dissertation combines experimental and analytic techniques (Chapters 2 and 3 respectively) to measure and to predict the performance of the A* best-first search algorithm, which solves path-finding problems defined in terms of finite strongly connected graphs. In this domain, we make numerous experimental performance measurements under systematically varying conditions, we derive general formulas in a simpler analysis model that purport to predict the experimental observations when evaluated at particular argument values that correspond to the experimental parameter settings, and we test the analytic predictions against the experimental observations. For reasons discussed subsequently, The A* experiments use as case study a randomly generated set of instances of the "Eight" puzzle of varying size (depth of goal).¹

Chapter 4 reports experimental results for a second problem domain, that of a class of satisficing assignment problems. Here we measure under varying conditions the performances of four functionally equivalent algorithms -- the so-called backtrack algorithm, a version of the so-called "network consistency" or constraint satisfaction algorithm of Waltz, and two new algorithms BACKMARK and BACKJUMP. The SAP experiments span four case studies: two sets of N-queens problems and two sets of randomly generated problems whose characteristics are specified by the values of certain parameters.²

The classes of problems of which the Eight puzzle and the N-Queens problems are elementary examples are defined broadly, and include many disparate problems, both simple and complex. Note that we are not interested primarily in the 8-puzzle or in the N-queens problems per se, but rather as relatively simple yet non-trivial case studies in which to explore general issues with rigor, principally the issue of predicting algorithm performance.

Most of the questions addressed in the dissertation concern the number of steps

¹ The Eight puzzle consists of eight tiles placed in a three by three board so that tiles may slide successively into the empty spot, forming a new tile configuration each time doing so. The objective is to find a sequence of tile moves transforming a given initial tile configuration into a given goal configuration. The 8-puzzle is depicted in the introductory section of Chapter 2.

² The N-queens problem is to place N queens on an N by N chessboard so that no two queens attack each other.

executed by an algorithm A when applied to a problem P, for various A and P and for various solution criteria, heuristics, and weighting coefficients, i.e., for various values of what we call problem specification parameters and control policy parameters.

The ability to predict, a priori and quantitatively, the performance of a given algorithm, when applied to a particular novel problem instance in its domain, is the central concern of this dissertation. Knuth addresses this concern succinctly:

"One of the chief difficulties associated with the so-called backtracking technique for combinatorial problems has been our inability to predict the efficiency of a given algorithm, or to compare the efficiencies of different approaches, without actually writing and running the programs." [Knuth 1975, p.121]

Knuth's contribution in that paper to improving the predictability of the backtrack algorithm as it is used in practice illustrates many of the issues that arise in the domain studied in Chapter 4, and the same issues arise in the A* domain, so let us review Knuth's results.

Knuth proceeds to define a model of a class of satisficing assignment problems (SAPs, as they are called in Chapter 4; our computational model is essentially the same as Knuth's). Based on a mathematical analysis, Knuth proposes a mechanical means to predict the number of nodes in the search tree produced by the backtrack algorithm when finding all solutions to an arbitrary SAP. The predictor, however, is not a closed-form mathematical formula, nor a non-closed-form formula, but rather a particular type of Monte Carlo experiment. The result of each experiment is an estimate of the number of nodes in the search tree, and Knuth proposes using the mean of the estimates over a number of iterations.

Because the values his procedure attempts to predict are mathematically well defined, it is at least conceivable that there exists a simple closed form formula of the same scope as Knuth's Monte Carlo predictor and of comparable accuracy. However, consider what arguments or parameters such a formula might take: in order to predict the performance of the backtrack algorithm for an arbitrary individual problem instance in the domain of the algorithm (the 8-queens problem, say), the values of the formula's parameters must distinguish each such problem instance from all others (e.g., from the 9-Queens problem, Instant Insanity, the Soma cube puzzle, Waltz' line drawing

interpretation problem, etc.). Since Knuth defined the domain of problems solvable by the backtrack algorithm very broadly (as we do also in Definition 4.1), one can easily suppose that simply defining an exhaustive set of distinguishing parameters and identifying the parameter values corresponding to a given problem instance may be problematic in itself. This is an essential point in distinguishing the two domains considered here from many others appearing in the analysis of algorithms literature.³

The point here is to contrast the backtrack algorithm with a sorting algorithm, say, for which a formula for the number of comparisons, say, can be given having a single integer parameter N , denoting the number of elements to be sorted. Such a formula does not predict the number of comparisons for individual permutations to be sorted, but predicts only the mean (say, or the maximum) of the number of comparisons over all permutations of a given number of elements N , i.e., over an ensemble of $N!$ problem instances. While prediction for ensembles of problem instances may prove satisfactory in the case of sorting, in contrast the mean performance of the backtrack algorithm in solving all satisficing assignment problems having N problem variables may not be an especially informative number. Hence the need for more problem specification parameters (or non-parametric means such as Knuth's) to distinguish one problem instance from all others in the class of problems constituting the domain of the algorithm, and whence the challenge in the tasks of formulating a computational model and analyzing the performance of an algorithm within that model.

In the two problem domains considered in this dissertation, the need for predictive abilities arises in practice typically when one must choose from among several algorithms, or heuristics, or values of a weighting parameter, or other control policy parameters, the candidate that will give the most efficient performance for the particular problem to be solved. Especially in domains in which for some problem instances an algorithm "will run to completion in less than a second, while other applications seem to go on forever" [Knuth 1975, p.121], it would seem that voluminous hard data, spanning as many independent conditions as possible, are desirable as a firm basis for assessing how a particular predictor might fare in a particular novel application.

³ See Weide [1977] for a survey; Knuth [1969], [1973a], [1973b] and Aho, Hopcroft and Ullman [1974] present examples in depth.

This introductory section has attempted to illustrate that the two problem domains considered in this dissertation have characteristics of interest both to AI and to analysis of algorithms research (although for not identical reasons), that these characteristics recommend a methodology that combines experiment and analysis in complementary and highly specialized and formalized roles, and that the richness of the domains make it difficult to obtain simply-stated general results that apply to individual problem instances as well as to ensembles of problem instances. The mathematical richness of these domains concomitantly permits, as we shall see subsequently, attempts to formulate certain elusive general concepts such as "knowledge" and "problem structure" in a strictly mathematical, albeit restricted, setting.

In a broad sense, the present results attempt to show that statements such as, "The problem of searching a graph has essentially been solved and thus no longer occupies AI researchers" [Nilsson 1974, p. 787], are premature.

1.2 Objectives, Methodology, and Scope

1.2.1 Objectives

Experiments are usually performed in order to verify, or sharpen or qualify or reject, a given hypothesis. We list now three such hypotheses about algorithm performance found in the literature that we submit to the test of hard data in subsequent chapters. Mackworth [1977] claims that Waltz-type constraint satisfaction or "network consistency" algorithms are "clearly more effective" than the backtrack algorithm for solving satisficing assignment problems. At the time of that claim, however, there was not a single numerical experimental result comparing the performance of the backtrack algorithm with that of a Waltz-type algorithm under strictly identical conditions (including identical problem instances and identical performance measures). Mackworth also claims that the number of steps executed by the backtrack algorithm "tends to" grow exponentially with the number of variables. The experimental data reported in Chapter 4 disagree with these conjectures in the cases tested.

Similarly, Nilsson, Pohl, and Vanderbrug conjectured that increasing the value of a weighting parameter W in A^* search will decrease the number of nodes expanded for a given heuristic function that estimates distance from the current node to the goal. In Chapter 2 we provide average case experimental evidence, supporting the conjecture under some conditions and disagreeing with it under other conditions; in Chapter 3, theorems under worst case tree search assumptions prove the conjecture false under some conditions and prove it true under other conditions.

In each of the above cases, a conjecture was stated in an overly general way, as if it was alleged to apply without exception to every problem instance in the domain of the algorithm. Since the classes of problems considered here are broadly defined and include widely disparate instances, intuition suggests that the conjectures are not universally valid, but rather are valid only for a subset of the problem domain. The results of the present experiments serve to delimit further the scope of the above conjectures.

Our experimental work was guided by certain other general objectives as well:

- 1) To determine the effect on the character of experimental results of (approximately) an order of magnitude increase in computer speed and main memory size, as compared with the machines available more than a decade ago when A^* search of the 8-puzzle was first investigated experimentally. In particular, extending the body of experimental data by a large factor can reveal new phenomena, i.e., instances in which the plotted performance measurement data show a visually apparent pattern whose existence was previously unsuspected.
- 2) To determine what practical applications can result from these experiments and analysis.
- 3) To amass a large body of experimental algorithm performance data as an end in itself, for the purpose of potentially stimulating further development of theoretical analysis in these domains, and so that the predictions resulting from such analysis may be tested conveniently against the observations compiled here.

Our mathematical analysis of A^* in Chapter 3 differs from others in that it is general enough to claim that the heuristic function is one of the independent variables, and in that the predictive applicability of this model is actually testable by direct experiment with problems and heuristic functions occurring in practice.

1.2.2 Methodology

Chapter 3 attempts to adhere to the standards of mathematical proof commonplace in the analysis of algorithms literature, so here we address only issues concerning the experiments in Chapters 2 and 4.

Experimental results about algorithm performance for particular cases are a poor substitute for analytically derived formulas of a more general scope, but can serve to guide the development of theory or suggest specific conjectures to prove, especially when general analysis is difficult. To insure that the experimental results are mathematically meaningful and can be compared with analytic predictions, we attempt to adhere to certain methodological standards: First, we define a precise computational model of experiments such that each datum observed by experiment is an estimate of the value of a particular mathematical function, evaluated at a particular set of argument values. Second, so that algorithm comparisons are meaningful, in all cases we execute the algorithms to be compared under identical conditions, including identical samples of problem instances and identical performance measures. We also report the precise conditions of the experiments (for the sake of reproducibility), and in many cases we count the number of distinct algorithm executions represented in a figure of plotted data (to indicate explicitly the extent of the data).

1.2.3 Scope

Chapter 2 defines a computational model for the A* best-first search algorithm for arbitrary problem graphs. The model defines several performance measures as functions of a state-space graph G , a heuristic function K , distance to the goal N (a measure of the size of the problem), and a scalar weighting coefficient W . We measure the values of these functions by Monte Carlo experiments over a randomly selected sample of 895 instances of the 8-puzzle of varying N , for each of three particular heuristic functions taken from the literature, and for each of eleven equidistant values of W . The results represent more than 26,000 distinct algorithm executions.

Chapter 3 analyzes a worst case mathematical model of A* assuming uniform trees in which there is a single goal node at level N . We give formulas for the number of nodes expanded as a function of N , of the branching factor M , of the estimate-

bounding functions $KMIN(i)$ and $KMAX(i)$ representing the heuristic function used as a control policy parameter to guide the search, and of a weighting coefficient W that serves as an additional control policy parameter.

In Chapter 4 we report the results of a set of performance measurement experiments comparing the so-called backtrack algorithm with an instantiation of a so-called Waltz-type "network consistency" algorithm and with two new algorithms, BACKMARK and BACKJUMP. Each of the algorithms is valid for a broadly and precisely defined class of satisficing assignment problems (SAPs) that includes numerous disparate familiar problems. The results span four functionally equivalent algorithms, three performance measures, two solution criteria, and four sample sets of SAPs, and the results represent more than 17,000 distinct algorithm executions. The four sample sets of SAPs include two sets of "N-Queens" problems (for N up to 50 in some cases) and two quite different types of randomly generated problems. One of the latter is a set of "random-N-Queens" problems whose members are constrained to be parametrically similar to N-Queens problems (i.e., to have the same size and "degree of constraint"). Results for this sample set (Section 4.4.2) simultaneously generalize the results for N-Queens SAPs to a set of "typical" problems, and determine how "typical" the N-Queens SAPs actually are. (See Section 1.4.3 for more detail.) The other sample set of randomly generated SAPs are identical in size but vary systematically in degree of constraint. The results in this case (Section 4.4.3) indicate how performance depends on degree of constraint, all other things being equal.

The results obtained are summarized in Sections 2.0, 3.0, and 4.0.

1.3 Mathematical Models: Defining Problems, Algorithms, Heuristics

In this dissertation the terms problem, algorithm, heuristic, degree of constraint, quality of solution, and others have particular mathematical definitions.

1.3.1 Problem Specification Parameters and Control Policy Parameters

The example of Knuth in Section 1.1 suggests that to predict algorithm performance for individual problem instances and individual variations or instances of

the algorithm in a context in which there are many such instances having disparate properties, a formula giving such predictions must have parameters distinguishing each problem instance and algorithm instance. In contrast with a simple sorting algorithm, for example, A^* is not a fixed algorithm but rather is an algorithm schema parameterized by a "heuristic distance-estimating function" that defines what is best during a particular invocation of A^* . To illustrate (without going into detail), we compare the following functions:⁴

<u>Model</u>	<u>Algorithm performance function</u>	
Quicksort	$C(N)$	(1-1)
"Median-of-k Quicksort"	$C(k, N)$	(1-2)
" ϵ -approximation schema"	$C(\epsilon, N)$	(1-3)
A^* for graphs (Chapter 2)	$X(G, K, W, s_r, s_g)$	(1-4)
A^* for graphs (Chapter 2)	$XMEAN(G, K, W, N)$	(1-5)
A^* for graphs (Chapter 2)	$XMAX(G, K, W, N)$	(1-6)
A^* for trees ("DEBET" - Chapter 3)	$XWORST(M, KMIN, KMAX, W, N)$	(1-7)
SAP-S (Chapter 4)	$T(S)$	(1-8)
SAP-N- k_i -L (Chapter 4)	$T(N, k_1, \dots, k_N, L)$	(1-9)

⁴ For those interested, Sedgewick's "median of k" version of the Quicksort algorithm [Sedgewick 1975, Chapter 8] is a generalized algorithm, instantiated for any particular invocation by specifying a value for k as an actual parameter to the procedure that codes the algorithm (see 1-2 above). So-called ϵ -approximation algorithm schemas have appeared in the literature of NP-complete problems [Garey & Johnson 1976]. An example is a travelling salesperson algorithm that finds a non-optimal tour, the length of which is bounded by the given value of ϵ [Karp 1976]. This schema is coded by a procedure whose formal parameter list includes a real-valued parameter representing ϵ (see 1-3 above). As in the case of Sedgewick's algorithm, the value of this parameter is freely chosen by the user from the set representing the domain of the parameter. Just as each value of k or ϵ , in the cases of Sedgewick and Karp respectively, determines one particular algorithm instance among those in the schema, so also each combination of values of KMIN, KMAX, and W determines one particular algorithm instance in the A^* schema.

As a notational device, we distinguish "problem specification parameter" and "control policy parameter", or p.s. parameter and c.p. parameter for short. We define the domain of a control policy parameter to be a set whose elements denote individual variations of a generalized algorithm. We shall thereby distinguish the analysis of A* in Chapter 3 from analyses of other algorithms by the number and dimensionality of the c.p. parameters. We define the domain of a problem specification parameter to be a set whose elements denote individual variations of a generalized problem, or individual problem instances. Hence in the examples listed above we distinguish the following:

<u>Algorithm performance function</u>	<u>Problem specification parameters</u>	<u>Control policy parameters</u>
C(N)	N	
C(k, N)	N	k
C(ϵ , N)	N	ϵ
X(G, K, W, s_r , s_g)	G, s_r , s_g	K, W
XMEAN(G, K, W, N)	G, N	K, W
XMAX(G, K, W, N)	G, N	K, W
XWORST(M, KMIN, KMAX, W, N)	M, N	KMIN, KMAX, W
T(S)	S	
T(N, k_1 , ..., k_N , L)	N, k_1 , ..., k_N , L	

Artificial intelligence researchers commonly refer to A* as a "heuristic" algorithm, because an instantiation of A* to solve a particular problem G may sometimes be caused to execute more quickly if supplied with a function of a certain sort used by A* to order the steps of the search, and because such a function is usually devised in practice by attempting to determine what special properties may hold for G. Here however, we treat a "heuristic" function of the sort used by A* as just another c.p. parameter (i.e., K in (1-4) and (1-5), KMIN and KMAX in (1-7); W is another c.p. parameter).

The scope of our A* analysis is thus somewhat akin to that of defining and analyzing a general computational model encompassing, say, all possible sorting algorithms of the sort that operate by comparing data elements and swapping their locations, characterizing each such algorithm instance by a mathematical function and deriving a general formula such that the performance function of Quicksort (say, or bubble sort) is obtained simply by plugging into the general formula the particular function that characterizes Quicksort.

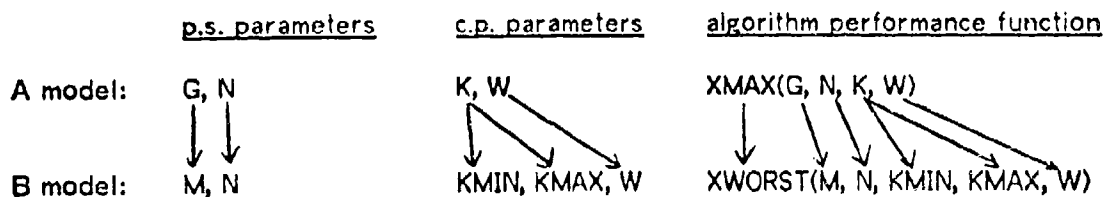
In analyzing a schema of algorithms parameterized by arbitrary functions, one might reasonably expect that formulas derived for the most general case are rather complicated, but that simpler and more intuitively meaningful formulas can be obtained if certain assumptions are imposed. This turns out to be the case in the current work (Chapter 3). The trick, of course, is to find the right assumptions.

We adopt the term algorithm schema here to denote formally the domain of a control policy parameter. In the case of multiple c.p. parameters, algorithm schema denotes the cross product of their respective domains. Hence an algorithm schema is equated with a set of algorithm instances. By this definition, Sedgewick's "median-of-k-Quicksort" algorithm is an algorithm schema on the odd positive integers; similarly, Karp's ϵ -approximation algorithm for the travelling salesman problem is an algorithm schema on the positive reals; Chapter 3 defines A* as an algorithm schema on the cross product of the real interval $[0,1]$ and the set of all pairs of functions of a certain sort.

In the same manner that an instantiation of c.p. parameters identifies an individual algorithm instance, so an instantiation of problem specification parameters identifies an individual problem instance, or an ensemble of problem instances. Functions (1-1), (1-2), (1-3), (1-5), (1-6), and (1-7) above illustrate a trivial use for this notation: N denotes a quantity representing the size of the problem of the sort defined in Section 4.1.1. For Quicksort, N denotes the ensemble of all permutations of a set of N elements. In the case of (1-7), a problem instance is a uniform tree having branching factor M and in which there is a single goal node at level N -- hence we have two p.s. parameters. In function (1-8), S is a p.s. parameter identifying a particular satisficing assignment problem. In (1-9), N , k_1 , ..., k_N , and L are p.s. parameters identifying the ensemble of all satisficing assignment problems having a particular size $(N, k_1, \dots, k_N)$ and degree of constraint (L) .

1.3.2 Analytic Predictions vs. Experimental Observations

To test analytic predictions against experimental observations when the analytic model is a simplification of the experiment model, we define a mapping from the experiment model to the analysis model. We noted in Section 1.3.1, for example, that we identify an A* experiment (in Chapter 2) by specifying particular values for its problem specification parameters G and N (i.e., for the ensemble of all problem instances of G of distance N), and for its control policy parameters K and W . Let us call this computational model the "A" model. The analytic model for A* in Chapter 3 (call this the "B" model) admits problem specification parameters M (positive integer) and N (non-negative integer), and control policy parameters $KMIN$ (a certain sort of function), $KMAX$ (a certain sort of function), and W (the real interval $[0,1]$). Hence to predict within the B model the outcome of an experiment in the A model we must map the particular values of the problem specification (p.s.) parameters in the A model to particular values of the p.s. parameters in the B model, and map the particular values of the control policy (c.p.) parameters in the A model to particular values of the c.p. parameters in the B model. This cross-model comparison can be depicted thus:



To identify the actual parameters for $XWORST$ corresponding to a given experiment in the A model, we map the given graph G , (e.g., the 8-puzzle graph), to a value M indicating the average branching factor of G ; N in the A model is mapped identically to N in the B model; a particular heuristic function K (e.g., the "number of tiles out of place" function) is mapped to particular functions $KMIN$ and $KMAX$; and W in the A model is mapped identically to W in the B model. Given such a mapping, a particular set of values for the p.s. and c.p. parameters of the A model determines two performance values: one for $XMAX$ and one for $XWORST$. The difference between these two values measures the accuracy of the analytic prediction of the experimental observation. Since the B model is a simplification of the A model, their comparison by this means permits an objective assessment of how realistic the assumptions imposed for tractability in the B model are.

Now note the cross-model comparison we make in the domain of satisficing assignment problems: a problem instance S in the experiment model (A model) is abstracted by a set of problem specification parameters $N, k_1, \dots, k_n, L$ in a simpler model (B model). The latter parameters define an ensemble of problem instances of which S is a member. Hence this cross-model comparison can be depicted thus:



In this case we predict algorithm performance for an individual problem instance in the A model (e.g., the 8-Queens problem) by an average case performance value in the B model. In Chapter 4 we estimate values in the B model using Monte Carlo experiment instead of analysis. So though we derive no analytic formulas for algorithm performance in this B model, our experiments estimate the values of such formulas for particular cases. Hence our experimental A model/B model comparison in this domain provides evidence about the accuracy of this B model in predicting corresponding algorithm performance in the A model. In so doing, we are attempting to test the restrictiveness of the B model assumptions in advance of obtaining results in that model.

1.3.3 Abstractions: Monotonicity Theorems on a Lattice of Algorithms

When modeling a number of algorithm instances, enumerated by the cross product of one or more control policy parameters, it is commonplace to determine how performance varies with the value(s) of the parameter(s). Hence, for example, Sedgewick [1975, Chapter 8] determines how the number of comparisons executed by "median-of-k-Quicksort" varies with the value of k (an odd integer). Similarly, one can determine how the performance of c -approximation algorithm instances vary with the value of c (a positive real) (e.g., [Karp 1976]). In Chapter 3, we seek to determine how the worst case cost of A^* tree search varies with the values of the c.p. parameters $KMIN$, $KMAX$, and W . In this case W is a real-valued scalar, but $KMIN$ and $KMAX$ are arbitrary functions from the natural numbers to the non-negative reals. There are no

obviously suitable total orderings on the set (called KB^* in Chapter 3) of all possible KMIN and KMAX functions, but we do note, however, the possibility of imposing a certain partial ordering (in fact an infinite continuous lattice) on KB^* . Then we prove monotonicity theorems, stating that if one function is less than another under the partial ordering, then its performance betters that of the latter under a similar partial ordering defined on the set of performance functions. In such manner we prove, for example, that under certain conditions the number of steps executed by A^* in the worst case grows monotonically with the relative error in the heuristic function's estimates of distance to the goal.

1.4 Examples of General Questions Modeled in a Restricted Context

We attempt to address within the restricted contexts of the dissertation simplified versions of several general questions of practical or theoretical interest. Examples are described in the subsections following.

1.4.1 How Much Does "Parameter Tuning" Change Performance?

Many problem solving systems (e.g., Samuel [1963], Hayes-Roth & Lesser [1977]) employ some sort of heuristic evaluation function to guide a search. Typically, the evaluation function incorporates a number of different terms, weighted differentially. Performance then varies with the relative weighting given the various terms. Choosing values for the weighting coefficients or parameters would be simplified in practice if the performance consequent to each possible setting could be predicted accurately a priori.

Our models of A^* in Chapters 2 and 3 assume a particular two term evaluation function whose terms are weighted by a single scalar parameter W . (One term measures distance from the root node of the search to the present node; the other term estimates distance to the goal node from the present node.) In Chapter 2 we measure performance as a function of N (size of the problem) for each of 11 equidistant values of W , and for each of three distinct heuristic functions (i.e., the heuristic function is one of the terms of the two term evaluation function). Chapter 3 derives formulas analytically for arbitrary values of W and heuristic function K , and we

compare the deduced analytic predictions to the experimental observations in Chapter 2. In this way, we obtain precise answers in a simplified setting of the general question. We treat another instance of the parameter tuning issue in analyzing the DEELEV algorithm in Section 4.2.3.

1.4.2 How Much "Knowledge" Buys How Much Performance?

It is conceivable that there may appear eventually a mathematical theory or theories about "knowledge" and its relation to problem solving performance. The experience of AI researchers with knowledge-based systems can be summarized by the statement "Expert knowledge buys expert performance" (see e.g., [Feigenbaum 1977]). Determining exactly how much knowledge buys how much performance is problematic, however, because a rigorous theory would require that "knowledge" and "performance" be well-defined, empirically measurable quantities. No precise definition of "knowledge" has yet been offered, even for a restricted domain. (In what units is "knowledge" measured? By what criteria can one decide whether an algorithm A possesses more knowledge than an algorithm B?)

Our approach offers no definition of "knowledge"; instead we simply present an informal interpretation of the results of Chapter 3 as if they constituted a particular type of theory about the relation of "knowledge" to algorithm performance in the A* domain. Chapter 3 derives formulas for worst case cost of A* as a function of the heuristic function used to guide the search (and of the weighting parameter W). Hence A* is considered in this interpretation as a general "knowledge engine", driven in a particular search by an arbitrary heuristic function encapsulating some "state of knowledge" about the problem to be solved. The analysis then determines the performance of the knowledge engine as a function of the state of knowledge it is given. This informal interpretation is presented in Section 3.7; the rest of Chapter 3 finds no use for the term "knowledge". Our interpretation of these analytic results as embodying a particular type of theory about the relation of "knowledge" to algorithm performance illustrates some of the mathematical subtleties of this elusive concept. This approach would permit a future comparison of two engines of comparable scope (e.g., A* and B* [Berliner 1978]) to see which one performs the better, given the same knowledge.

1.4.3 How to Measure "Structure" in a Problem?

Each problem has some individual or characteristic "structure" that distinguishes it from other problems, but such a statement conveys little information in itself. Like "knowledge", problem "structure" is a term that is easier to talk about informally than to define precisely. In what units is "structure" measured? What might it mean to say that problem A has "more structure" than problem B? In what way is "structure" related to algorithm performance?

Our approach to measuring the dependence of algorithm performance on problem "structure" in Section 4.4 requires no formal definition of the term, because we do not attempt to measure structure in a problem directly. Instead we measure observable manifestations of its presumed existence. Our approach assumes that "structural" differences between two problems are reflected as differences in the performances of a given algorithm in solving both problems. Specifically, we apply the backtrack algorithm to the 8-queens problem, and then apply the same algorithm to a set of randomly generated problems that are parametrically similar to the 8-queens problem (in size and degree of constraint). The difference in performance between the 8-queens problem and the "random-8-queens" problems reflects the difference between a particular structure and random structure, size and degree of constraint being equal. This experimental approach is then generalized to a comparison between N-Queens problems and "Random-N-Queens" problems (for various values of N); we also generalize to using three other algorithms in turn in place of the backtrack algorithm (to see whether the algorithms react differently to the existence of structure in the problem); we also generalize to several performance measures.

Note that the present approaches to investigating "heuristic knowledge" and "problem structure" share a general characteristic: we do not attempt to measure "heuristic knowledge" or "problem structure" directly; instead we measure observable manifestations of their presumed existence, in terms of algorithm performance.

1.5 Tradeoffs: Why These Experiments, Why This Analysis?

Within the technical context described in the preceding sections, we could have chosen other experiments and other analyses different from those reported in

subsequent chapters. The reader will doubtless think of numerous interesting possibilities. Accordingly, it seems useful to provide rationales for the choices we made. Rather than attempt to justify our choices, we simply report the possibilities considered, and the criteria on which the selections were based.

Experiments: few runs on many problems vs. many runs on few problems

We have opted to obtain extensive and detailed experimental results on a relatively small number of problems. This facilitates determining what new algorithm behavior phenomena can be observed given more computing than was previously available to produce such phenomena. However, we do apply this detailed experimental approach to two problem domains, thus obtaining some breadth as well as depth in the results.

Generating data vs. analyzing data

Our technical objective in the experiments is simply to obtain the performance values plotted in the figures. Careful and rigorous quantitative analysis of the experimental values is beyond the scope of the present work. Just as we insist on having hard data as the referent of any statement about algorithm performance, so also we insist on rigorous explanations or none at all. Hence we eschew any attempts to "explain" the visually apparent patterns in the data, except by theorems within a formal model, as in Chapter 3. Hence we provide a large body of quantitative data (tabulated in Appendix C) against which to test future conjectures and mathematical theories, and to serve as the subject of a subsequent detailed analysis.

Level of detail

In addition to choosing particular subjects for experiments and analysis, we also had to choose the amount of detail to be pursued for each subject. Our choices reflect the general point of view that so little is known about the computational properties of the present search algorithms, that breadth as well as depth of results is desirable. That is, the marginal value of new knowledge is sometimes greatest where little or none exists. Hence the desire for more detail in one section of this dissertation was sometimes traded off against the possibility for some detail in another. A number of such possibilities for future work are enumerated in Appendix B.

In addition, a number of important issues were completely excluded as beyond the scope of the present work. Included among these are: relating the efficiency of a given A* heuristic function to the cost of computing it, and to the memory size required to implement it; the cost-effectiveness of the analytic predictions of A* performance for the 8-puzzle heuristics given in Section 3.5; and symmetry or representation issues in search of satisficing assignment problems. Possible extensions of the present work concerning these issues are listed in Section 6.2.6.

These various tradeoffs reflect the exploratory nature of this thesis: to obtain results of various sorts in each of two problem domains, and to obtain both experimental and analytic results.

1.6 A Note on Reading this Dissertation

[Nilsson 1971, Chapter 3], [Weide 77], and the survey portion of [Mackworth 1977] are the most concise general background references for this dissertation, covering respectively the areas of state space search (relevant to Chapters 2 and 3), methodology of analysis of algorithms (Chapter 3), and a comparison of backtrack vs. Waltz-type algorithms for satisficing assignment problems (Chapter 4). Although the dissertation attempts to be self-contained, some familiarity with these sources is useful. For the benefit of readers with particular interests, each of Chapters 2, 3, and 4 attempts to be more or less self-contained, and may be read independently of the others. Toward this end, Sections 2.0, 3.0, and 4.0 summarize the technical results of the dissertation. Each of these chapters also contains a section concerning conclusions and future work. In addition, a condensation of Chapter 2 (with highlights of Chapter 3) has appeared [Gaschnig 1977a]. Similarly, condensations of Chapter 4 have also appeared [Gaschnig 1977b, 1978].

Appendix A gives a glossary of terms and symbols used in Chapters 2, 3, and 4. Appendix B enumerates a number of immediate extensions of the experiments and analysis of the dissertation. The experimental results reported in this dissertation consist of various sets of numbers, ordered pairs of numbers, and so on. In the main body of the text, most of these appear in the form of plots instead of in tables of numbers, both for the sake of making more apparent the relations between the plotted

values, and so as not to interfere with the flow of the text. Appendix C tabulates each value plotted in most of the figures of Chapters 2 and 4 and Section 3.5.

This dissertation is directed toward both readers familiar with artificial intelligence research and readers familiar with analysis of algorithms research. The style of the text reflects an attempt to communicate the results to the union of these two sets of readers, rather than to their intersection. Consequently, the text presumably does not necessarily satisfy "all of the people all of the time". Hence some readers seeking only mathematics may find some supplementary comments, explanations, and conjectures to be superfluous. In some cases such diversions serve a purely pedagogic end; in others, they represent attempts to say something useful, even if imprecisely, where adherence to strict precision would permit nothing at all to be stated. Conversely, other readers may find the level of detail and guarded conclusions (based only on hard data or theorems) contrary to a desire for general statements.

Section 1.1 suggests the viewpoint that a measure of our understanding of the performance of search algorithms is the ability to predict a priori the performance as measured by experiment. Accordingly, some readers may find it interesting, when the text cites a figure of plotted data, to spend a moment before looking at the figure in an attempt to decide what they expect the plotted curves to show.

CHAPTER 2

EXPERIMENTAL PERFORMANCE MEASUREMENT OF A^* :
A CASE STUDY WITH THE "EIGHT" PUZZLE¹

2.0 Summary of Chapter

This chapter attempts by Monte Carlo experiments to extend our understanding of the performance of the A^* best-first search algorithm. We define several performance measures of A^* , as functions of a problem graph Q , a heuristic distance-estimating function K , distance to the goal N , and a scalar weighting coefficient W , then measure the values of these performance functions experimentally over a sample set of problem instances. In this case study, Q is fixed (the 8-puzzle) and the other parameters vary, so that the values of each performance measure are plotted for each of three K functions as a function of N and W . The data represent more than 26000 distinct executions of A^* , taken over 895 problem instances of varying values of N .

The data suggest that one of the three heuristic functions subjected to experiment beats the "exponential explosion" in cost with size of the problem, and that the other two do not (at least over the range of N tested). But the latter can be made to do so simply by giving more weight (W) to the heuristic-estimate-of-distance-to-goal term (K) in the evaluation function and less weight to the distance-from-root term. However, this reduction in the average number of nodes expanded (i.e., in the values of $XMEAN(N)$) occurs only if N is large with respect to the maximum value, which equals the diameter of the graph to be searched. For "medium-sized" N , $XMEAN$ actually increases with W , suggesting the possible efficacy of varying W dynamically. A limit is observed to the effects of adjusting W : the performance of the best-performing of the three heuristic functions does not change after W is increased beyond a certain value, whereas the other two functions show improvement over the entire range of W . Also, if K_i has smaller $XMEAN(N)$ than K_j for one value of W , then the same ordering is observed to hold for each other measured value of W ; this has potential practical implications for using experimental results for "small-sized" and "medium-sized" problem instances as predictors of relative performance for "large-sized" instances.

Increasing W also increases the lengths of the solution paths found, but unexpectedly, for large W faster heuristics find shorter solutions, whereas for smaller W this is not always the case. In other words, we observed by inspection for large N that solution quality can be traded for speed by changing W (holding K fixed) but not

¹ The bulk of the experimental data reported in this chapter appeared first in [Gaschnig 1977a].

by changing K (holding W fixed). For "medium-sized" N , increasing W beyond a certain value brings worse solution quality at greater cost.

Also, the frequency with which A^* "hops around" the search tree during search is related to XMEAN performance, in a pattern common to all three K functions that exhibits three distinct phases in functional dependence on N . The absolute frequency of "hops" is so high for one of the three K functions as to suggest the possibility that ordered depth-first search (for which the length of the solution path equals the number of nodes expanded) expands fewer nodes than A^* when using the same K function. Also, for two of the three K functions, the number of nodes expanded that occur at level i of the search tree increases "exponentially" to a level representing about half the distance to the goal node, then decreases at about the same rate for higher levels. For the best performing of the K functions, on the other hand, the distribution of nodes at level i is flat, consistent with XMEAN performance.

We support each of the above claims individually with numerical results.

To illustrate how such results might serve as predictors of performance in more complex systems, conjectures supported by the current data are interpreted as if they applied to a best-first implementation of the program construction phase of the PSI program synthesis system.

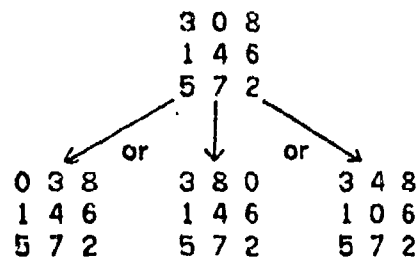
Since performance is functionally dependent upon the distribution in the distance-estimate values computed by a K function, these distributions are determined experimentally for each K function. To determine how sensitive these approximations are to the number of samples on which the estimates are based, we obtain distinct approximations for two different sample sets, one having more than ten times as many samples as the other. The results show that the two samples yield identical estimates in all but a few cases. The approximations show that the "bandwidth heuristics" assumption upon which Pohl's worst case cost model of A^* is based is not realistic for the 8-puzzle heuristic functions tested here. This motivates an attempt to relax the restrictions of the "bandwidth" model, which is the subject of Chapter 3. The observed worst case performance (XMAX) data collected during the experiments reported in this chapter constitute the values that the analytic worst case formulas derived in Chapter 3 purport to predict.

He therefore who wishes to rejoice without doubt in regard to the truths underlying phenomena must know how to devote himself to experiment.

Roger Bacon

2.1 Introduction

This chapter reports numerical measurements of the performance of the A* best-first search algorithm under conditions varying the heuristic function used to guide the search, the depth of goal, and the value of a weighting parameter, representing in all some 26,850 distinct algorithm executions, using as a case study a sample set of 895 instances of the "Eight" puzzle (hereafter denoted "8-puzzle"). The 8-puzzle [Schofield 1967] is a one-person game whose objective is to rearrange a given initial configuration of 8 tiles on a 3x3 board into another given goal configuration by repeatedly sliding a tile into the orthogonally adjacent empty location, like so ("0" denotes the empty location):



Some of the difficulties in building complex AI performance systems arise from an inability to predict performance a priori. Suppose in designing such a system that two alternative heuristics for doing the same task have been proposed: Which will give better performance, heuristic A or heuristic B? Debate is sometimes avoided by using both in a multi-term evaluation function, if the system in question uses an evaluation function of some sort to guide behavior. But performance then depends on how much weight each term is given. These prediction questions -- which heuristic is better? what weighting value is best? -- constitute the focus of the experiments reported

here. The premise of the approach taken here is that attempts to understand more completely the behavior of the A^* algorithm will benefit from the existence of data measuring its behavior over a range of conditions.

The experimental results reported in the following sections of this chapter differ from those of previous experiments with the 8-puzzle or 15-puzzle ([Doran & Michie 1966], [Michie 1967], [Doran 1968], [Michie & Ross 1970], [Rendell 1977]) in: a) volume of data collected, giving greater statistical significance over a range of three heuristic functions, eleven values of a weighting parameter, and several performance measures; b) new measures of "internal" behavior during search; c) measurements of the error in the heuristic distance-estimate values. The latter are used as particular argument values to the analytic formulas derived in Chapter 3; hence we test the analytic results of Chapter 3 by comparing their predictions for the case of the 8-puzzle with the experimental observations obtained during the experiments reported in this chapter.

The A^* algorithm [Hart, et al. 1968, Nilsson 1971] is an example of a relatively simple mechanism that shares certain similarities with more complex mechanisms that occur in practice. Several existing complex AI performance systems use some sort of best-first scheduler to decide what to do next, examples of which include the HEARSAY-II speech understanding system [Hayes-Roth & Lesser 1977], the HWIM speech understanding system [Woods 1976], and a chemical compound synthesis program [Powers 1975]. Section 2.6 makes an explicit analogy between simple search and more complex search.²

The A^* algorithm embodies the idea of a best-first search. In basic terms, there are at any given time a finite set of discrete options of what to do next, and choosing

² While the possibility of demonstrating a connection between simple search and complex search may serve in part to motivate the experiments reported here, the present results are limited to a case study of A^* search of the 8-puzzle. That is, here we measure the performance of A^* under varying conditions, leaving to future work the experimental measurement of more complex systems in a manner that permits quantitative comparison with the present results. Note that we are not interested in the 8-puzzle per se, but as a problem that is relatively small and convenient to manipulate, yet exhibits interesting phenomena (this to be demonstrated) that may hold for a broader class of problems.

and executing an action results in a new set of options: the still unchosen ones plus new ones generated by performing the chosen action. If there is no obvious way to totally order these actions in advance (and recall that some don't exist until others are performed), then one approach is to assign a number to each potential action as it appears, according to how good it might be to do that one, independently of any other actions that may have been executed already. Then an effective general method is to start with the set of initial options, and choose iteratively the action that has the smallest value (smallest is best) until the specified goal condition is satisfied. The A^* algorithm schema operates on this principle, using an arbitrary ordering function $F(s)$ to solve problems like the 8-puzzle. The Graph Traverser [Doran & Michie 1966] and the HPA algorithm schema [Pohl 1970a] are essentially the same as A^* .

The 8-puzzle can be modeled exactly as a collection of points (tile configurations) and lines connecting them (tile moves), i.e., as a graph. In the 8-puzzle graph, each of the $9!$ nodes represents a distinct tile configuration, and an edge connects two nodes if and only if the corresponding tile configurations differ by a single tile move. In general, we define a problem graph to be any finite, directed, strongly connected graph Q having no loops and no parallel edges.³ The 8-puzzle is an undirected graph since every move has an inverse. The prohibition against loops and parallel edges in the graph model of a state space problem is quite natural: it is typically irrelevant whether or not a state (e.g., a tile configuration in the 8-puzzle) is connected to itself, or to distinguish between single and multiple connections between two states. In general, a scalar value may be associated with each edge, representing the cost of traversing that edge. Here, we assume the edges of the 8-puzzle graph to have unit weight. Figure 2.1-1 shows a portion of the 8-puzzle graph.

2.2 Cost and Solution Quality for 8-Puzzle Heuristic Functions

³ Throughout, formally defined terms are either underlined or set off from the text. Definitions of graph theoretic terms such as "strongly connected" and "loop" and "parallel edges" appear in [Busacker & Saaty 1965]. A graph is strongly connected if there is a path from any node to any other node. Actually, the 8-puzzle graph is not strongly connected, but rather consists of two disconnected components (each of which

Best-first search guided by heuristic knowledge can be more efficient than breadth-first search, but how much more efficient? Under what conditions do heuristics beat the "exponential explosion" that besets breadth-first search? To motivate the somewhat extended definition of a computational model that, first consider Figure 2.2-5, which shows the mean number of nodes expanded as a function of the distance to the goal for three particular heuristics for the 8-puzzle. The apparent qualitative difference between one of the curves and the other two is of particular interest: could this have been predicted a priori?

The objective of the search is to find a simple path in the graph from one specified node to another, in the case of the 8-puzzle to find a sequence of moves of the tiles transforming one tile configuration into another. Each possible choice of initial or root node s_r and goal node s_g in a problem graph Q defines a distinct problem instance (s_r, s_g) , hence a problem graph Q having V nodes induces a set $U(Q)$ of V^2 problem instances.⁴ The minimum distance in Q between any two nodes s_i and s_j is always defined since the graph is strongly connected, and is denoted $h(s_i, s_j)$. Search of a particular problem instance (s_r, s_g) finds a solution path in the graph from s_r to s_g whose length equals or exceeds $h(s_r, s_g)$. For brevity, we will say "a problem instance (s_r, s_g) of distance N " to mean "a problem instance (s_r, s_g) such that $h(s_r, s_g) = N$ ".

Many common puzzles satisfy these formal conditions exactly (e.g., [Nilsson 1971, pp. 39-41, 77-78], [Jackson 1974, pp.81-84, 110-115], [Raphael 1976, pp. 79-86], [Wickelgren 1974, pp. 49-57, 78-80 cf.]). Somewhat less frivolous examples are certain algebraic manipulation problems [Doran & Michie 1966, pp. 254-255], [Doran 1967, pp. 114-115] and a version of the travelling salesperson problem [Doran 1968], [Harris 1974]. Other problems have state space models that are more complex but basically similar, e.g., search for connection between two concepts (i.e., nodes) in a semantic network (i.e., graph), and those mentioned in the introduction to this chapter.

is strongly connected); for our purposes we consider search within one such component.

⁴ Since the 8-puzzle graph consists of two disconnected components, we include in $U(Q_{8\text{-puzzle}})$ exactly those problem instances (s_r, s_g) for which s_r and s_g belong to the same component. Hence the cardinality of this set is $2(9!/2)^2 \sim 6.6 \cdot 10^{10}$.

From among these candidates, the 8-puzzle was selected for this case study because: a) extant experimental results (cited earlier) motivate and provide contrast for the current results; b) the 8-puzzle is still "unsolved" in the sense that no optimal A* heuristic or other sort of algorithm for the problem is known, nor have the performances of the known A* heuristics been analyzed or adequately measured experimentally; and c) its graph, having $9!$ nodes, is sufficiently complex to be a source of interesting phenomena (this to be demonstrated).

Algorithm A* can be used to solve any problem instance (s_i, s_g) of any problem graph Q .⁵ Heuristic selectivity in A* search is obtained by evaluating, at each node s encountered, a function $F(s)$, always choosing next a node with smallest F value from among those already evaluated.⁶ Any function $F: U(Q) \rightarrow \mathbb{R}^+$ is permitted, where $\mathbb{R}^+$ throughout denotes the non-negative reals; the efficiency of the resulting search depends on the properties of F . A problem graph is typically specified in practice by a successors function: $SUC(s)$ denotes, for any node s , the set of nodes v_i for which there exists an edge from s to v_i . Typically, $SUC(s)$ is implemented by a set of operators, each of which transforms a given state s into another state v_i , provided the operator's precondition is satisfied. In the 8-puzzle context, one such operator might have the effect of moving the hole upward if it is not in the top row of the board.⁷

⁵ Our treatment of A* attempts to be self-contained; [Nilsson 1971, pp. 43-79] is an excellent reference.

Algorithm A*:

1. Mark s_r as "OPEN" and compute $F(s_r)$.
2. Choose an OPEN node s whose F value is minimal, resolving ties arbitrarily but always in favor of a goal node.
3. If s is a goal node, then terminate.
4. Mark s as "CLOSED", compute $SUC(s)$, and compute $F(v_i)$ for each successor node v_i of s . Mark each such node as OPEN if it is not already marked CLOSED. Remark as OPEN any CLOSED node v_i whose F value is smaller now than it was when it was marked CLOSED. Go to step 2.

An execution of step 4 expands node s . Since A^* searches graphs, not just trees, step 4 tells what to do if more than one path to a given node is found. Typically, back pointers are used to record the path to the root node from any node node in the search tree (including, eventually, the goal node). Figures 3.6 and 3.8 in [Nilsson 1971, pp. 57,67] illustrate A^* search for the 8-puzzle, using two of the three heuristic functions studied here. Comparison of the latter figures with Figure 2.1-1 indicates a large reduction in number of nodes expanded using A^* search, as compared with breadth-first search.

As noted above, F may be any function from pairs of nodes of Q to the non-negative reals; in this thesis, however, we restrict attention to a particular form examined in [Pohl 1970a], [Pohl 1970b], [Munyer & Pohl 1976], [Munyer 1976], [Vanderbrug 1976], namely

⁶ Note that what we call F , [Hart et al. 1968] call f , as do most or all other reports about A^* . Similarly we denote by G what these others call g , and by K what these others call h . Footnote 9 offers a rationale for this departure from conventional notation.

⁷ The definition of a problem graph by a set of operators contrasts with the assumptions of [Dijkstra 1959] and [Tarjan 1975], in which the graph is input as a connection matrix, an edge list, or other similar scheme. Encoding the 8-puzzle in such manner would require a large amount of storage. As seen by comparing A^* search with the algorithms discussed by Dijkstra and Tarjan, the characteristics of the problem of finding a path in a graph depend strongly on the way the graph is represented.

$$F(s) = (1-W) G(s) + W K(s) \quad (2.2-1)$$

where $K: U(Q) \rightarrow \mathbb{R}^+$, W is a real such that $0 \leq W \leq 1$, and $G(s)$ denotes the distance in the search tree from s_r to s .^{8,9} If $K(s)$ is interpreted as an estimate of $h(s)$, then $F(s)$ as in (2.2-1) is a linear combination of the distance in the search tree from the root node s_r to the current node s and the heuristic estimate of distance from s to the goal node s_g . Informally, K contains the knowledge or information about Q available to guide the search. Note that K is a function of two nodes of Q (i.e., current and goal), but for simplicity we write $K(s)$ instead of $K(s, s_g)$ when goal node s_g is implicit (as it is during a given search). The degenerate case $W = 0$ or $K(s) = 0 \in K_0(s)$ corresponds to a breadth-first search, and is considered here only for purposes of comparison. The reader may verify that A^* terminates for any $F(s)$ satisfying (2.2-1), provided that $0 \leq W < 1$.

⁸ The form in (2.2-1) generalizes the form $F(s) = G(s) + K(s)$ studied in [Doran & Michie 1966], [Doran 1967], [Doran 1968], [Hart, et al. 1968], [Chang & Slagle 1971], [Nilsson 1971], [Martelli 1977], [Gelperin 1977]. A dynamic weighting form that generalizes (2.2-1) is investigated in [Pohl 1977].

⁹ Note that what we call K , [Hart et al. 1968] call $\hat{h}$, as do most or all other reports about A^* . The $\hat{h}$ notation may suggest to some the role of the heuristic function as an estimator of another function, namely, the function $h(s_i, s_j)$ that gives the exact distance between arbitrary nodes s_i and s_j in the graph. Our change in notation reflects a minor point of emphasis in the present work, namely, that a K function can be any function from pairs of nodes in the graph to the non-negative reals. In Chapter 3 we derive results concerning such a set of K functions. One important question is whether good performance is restricted to heuristic functions that are accurate estimators of the distance to the goal node. Other notation used in this chapter and in Chapter 3 (e.g., KMIN and KMAX functions) would be awkward to express consistently with the $\hat{h}$ notation.

Similarly, in other reports on A^* $\hat{g}(s)$ denotes the minimum distance from the root node s_r to node s found during the search, so that $\hat{g}(s)$ is an upper bound on $g(s)$, the actual minimum distance in the graph from s_r to s . In these other reports, $\hat{f}(s)$ is a linear combination of $\hat{g}(s)$ and $\hat{h}(s)$, and $f(s)$ is a linear combination of $g(s)$ and $h(s)$. For consistency, we denote by G what these others call $\hat{g}$, and by F what these others call $\hat{f}$. To be consistent, we really should use H instead of K to denote what these others call $\hat{h}$. Our use of the symbol K is simply mnemonic for "knowledge": the heuristic function encodes or represents some knowledge or information about the problem graph. Of course, no substantive issue is connoted by the present minor departure from the conventional notation.

Note the special cases $W = .5$ and $W = 1.0$ of (2.2-1). In the case of $W = 1.0$, (2.2-1) reduces to $F(s) = K(s)$, i.e., the distance estimate term alone. In the case of $W = .5$, (2.2-1) is equivalent to $F(s) = G(s) + K(s)$ in that they order identically the nodes expanded during the search. The equivalence arises because the precedence ordering between two nodes s_i and s_j is dependent on the relative values of $F(s_i)$ and $F(s_j)$, not on the absolute values of these expressions. This and the following two sections assume $W = .5$. Section 2.3 considers eleven values of W spanning its range from 0 to 1. Section 3.6 in this thesis and [Pohl 1970a, Pohl 1970b, Nilsson 1971, Vanderbrug 1976] motivate the study of the form given by (2.2-1), for reasons relating to the possible effect of the presence of the $G(s)$ term in providing "insurance" against excessive search.

We consider three K functions for the 8-puzzle taken from the literature [Doran & Michie 1966], [Nilsson 1971].

$K_1(s)$ = the number of tiles that occupy a board location in s different from the location occupied by that tile in the goal node s_g .

$K_2(s)$ = the sum, over all 8 tiles in s , of the minimum number of moves required to move the tile from its location in s to its desired location in s_g , assuming that no other tiles were blocking the way.

$K_3(s) = K_2(s) + 3 * seq(s)$,
where $seq(s)$ counts 0 if the non-central squares in s match those in s_g up to rotation about the board perimeter, and counts 2 for each tile not followed (in clockwise order) by the same tile as in the goal node.

The coefficient value 3 in the definition of K_3 was suggested as favorable by credit assignment experiments in [Doran & Michie 1966].

For given values of Q , K , W , and (s_r, s_g) , we define the cost of search and the goodness of the solution found as follows.

Definition: Cost and Solution Quality

$X(Q, K, W, s_r, s_g)$ denotes the number of executions of step 4 of A^* before search terminates, for the case of problem instance (s_r, s_g) using heuristic function K and weight value W .

$P(Q, K, W, s_r, s_g)$ denotes the length of the solution path found under the same conditions.

$$L(Q, K, W, s_r, s_g) = P(Q, K, W, s_r, s_g) / h(s_r, s_g)$$

Note: We will conveniently drop arguments from formulas when their values are known implicitly. For example, we write $X(K, s_r, s_g)$ in place of $X(\text{Q8-puzzle}, K, .5, s_r, s_g)$ when it is given that $Q = \text{Q8-puzzle}$ (as in the remainder of this document) and $W = .5$ (as in this section and in sections 2.4 and 2.5).

Note that $X(s_r, s_g) \geq P(s_r, s_g) \geq h(s_r, s_g)$ by definition. We say that a particular search is optimal if and only if only nodes along the solution path are expanded and the solution path found is of minimal length, i.e., if and only if $X(s_r, s_g) = P(s_r, s_g) = h(s_r, s_g)$. In general, $X(s_r, s_g)$ may greatly exceed $P(s_r, s_g)$, and $P(s_r, s_g)$ may greatly exceed $h(s_r, s_g)$. Note also that L expresses solution quality as a fraction of the minimal length of the solution path found for a problem instance, so that by definition $L \geq 1$, with equality if and only if a minimal length solution is found.^{10,11}

Theory tells us ([Hart et.al. 1968], [Pohl 1970b]) that for any Q and for $W \geq .5$ if $K(s) = h(s)$ for all nodes s (i.e., if K is the perfect estimator of h), then for all problem instances $(s_r, s_g) \in U(Q)$, it is the case that $X(Q, K, W, s_r, s_g) = h(s_r, s_g)$.¹² The curves labeled "optimal" in the figures of this chapter take these values.

¹⁰ The statements " $X(s_r, s_g) \geq h(s_r, s_g)$ " and " $L \geq 1$ " illustrate another type of circumstance in which we suppress the appearance of arguments, namely when the statement holds over all values of the omitted arguments.

¹¹ This definition of L is analogous to the measure of the goodness of non-minimal solutions that are found by a certain algorithm for certain restricted types of traveling salesperson problems [Karp 1976]. This algorithm schema finds tours whose lengths exceed the minimal length by a factor guaranteed (probabilistically for large N) not to exceed $1 + \epsilon = L$. The latter result is an example of a growing body of related complexity analyses of so-called epsilon approximation algorithm schemas for NP complete problems [Johnson 1974], [Garey & Johnson 1976], [Weide 1977 pp. 305-309].

The results of previous mathematical analyses of A^* permit little to be predicted about the performances of these three particular K functions in solving arbitrary problem instances of the 8-puzzle. The A^* admissibility theorem [Hart et.al. 1968] states that if $K(s_i, s_j) \leq h(s_i, s_j)$ for all $(s_i, s_j) \in U(Q)$, then $L(s_r, s_g) = 1$ for all $(s_r, s_g) \in U(Q)$. Hence we conclude that $L = 1$ for K_1 and K_2 if $W \leq .5$. Since K_3 does not satisfy the condition of this theorem, nothing can be deduced formally about its L values.

Regarding the X measure, formal theory [Hart et.al. 1968], [Pohl 1970a], [Pohl 1970b], [Nilsson 1971], [Harris 1974], [Vanderbrug 1976], [Pohl 1977] tells us only that K_2 and K_1 never expand more nodes than does breadth-first search, provided that $W \leq .5$. We cannot even deduce from the A^* optimality theorem ([Hart, et al. 1968], [Gelperin 1977]) that $X(K_2) \leq X(K_1)$ always; to apply that theorem it is required that one heuristic function's estimate be always greater (as opposed to greater than or equal to) another's.

Previous experiments revealed certain phenomena, but the data were too limited to permit very precise generalizations. For example, as a rationale for preferring K_3 to K_2 , Nilsson theorizes that

"Often heuristic power can be gained at the expense of giving up admissibility by using for $[K]$ some function that is not a lower bound on h ." [Nilsson 1971, p. 66]

A rigorous answer to the question thus posed by Nilsson requires precise definitions for his terms. As possible definitions for "heuristic power", Nilsson proposed the "penetrance" and "effective branching factor" measures introduced by [Doran & Michie 1966]. Since Nilsson's statement may not hold for every possible choice of Q , (s_r, s_g) , and K , it is interesting to determine for which choices it holds, and for which it does not. Also, it is interesting to measure the amount of gain in "heuristic power", if any, that is realized. Translating to the present formalism, we measure heuristic power

¹² Gelperin [1977] points out that for this to be true it is also necessary that ties among nodes having equal $F(s)$ values must be resolved in favor of the one having the smaller $K(s)$ value.

The conflicting evidence described above -- a specially chosen good example vs. a specially chosen bad example -- makes evident the need for measuring performance in the aggregate, over a set of problem instances, all having the same minimum distance, N , from initial node to goal.¹⁴ Hence:

Definition: Aggregate performance measures

$XMEAN(Q, K, W, N)$ denotes the simple arithmetic mean of the values of $X(Q, K, W, s_r, s_g)$ over all problem instances (s_r, s_g) in the set $U(Q)$ such that $h(s_r, s_g) = N$. (Note that N ranges over the non-negative integers not exceeding the diameter of Q .)

$XMAX(Q, K, W, N)$ and $XMIN(Q, K, W, N)$ are defined similarly.

$LMEAN(Q, K, W, N)$ is defined in terms of $L(Q, K, W, s_r, s_g)$ exactly as $XMEAN$ is defined in terms of X .

$LMIN(Q, K, W, N)$ and $LMAX(Q, K, W, N)$ are defined similarly.

Figure 2.2-1 shows the result of measuring X (by executing the search using heuristic function K_1 and $W = .5$) for a set of 895 randomly chosen problem instances of the 8-puzzle (from a population of about $6.6 \cdot 10^{10}$ possible instances). (Here randomly means independently, with replacement, and approximately uniformly; details of the selection procedure are given in Section 2.7.) The instances are grouped on the abscissa according to the actual minimum distance $h(s_r, s_g)$ between initial node s_r and goal node s_g . For $N = 10$, for example, 40 problem instances such that $h(s_r, s_g) = 10$ were generated randomly, and a measurement was made of the value of $X(K_2, .5, s_r, s_g)$ for each of these instances. The mean of these 40 experimentally measured values is plotted as $XMEAN(K_2, .5, 10)$. The samples are distributed with respect to N as follows: there are 40 samples for each of $h(s_r, s_g) = N = 2, 3, \dots, 20$; 30 samples for $N = 21, 22, 23$; 25 samples for $N = 24$; 12 for $N = 25$, and 8 for $N = 26$. This sample set is used in each experiment reported in this chapter. The maximum value of N for the 8-puzzle (i.e., the diameter of the graph) is 30 [Schofield 1967].¹⁵ Note that this sample set represents about 10^{-8} of the total population of problem

¹⁴ In practice, the value of N is not known a priori. For any particular problem instance, however, N has some definite value. Our objective is to determine A^* performance as a function of N : if the value of N happens to be such and such, then the number of nodes expanded is thus and so.

instances for the 8-puzzle. The procedure used to generate this sample set of problem instances is described in Section 2.7.

The vertical bars on the XMEAN curve in Figure 2.2-1 (at $N = 10, 15$, and 20)¹⁶ and in subsequent figures in this and following chapters measure twice the standard deviation of the sample XMEAN (one above and one below the value of XMEAN). This is a statistical measure of how accurately the experimentally measured value of XMEAN approximates the true value of XMEAN [DeGroot pp. 185-186, 350-351], [Drake 1967, p. 206].

Specifically, the true value of XMEAN is expected to fall within the indicated interval at the 67% confidence level, and within twice this range at the 95% confidence level. This assumes that the X values are selected independently from a normal distribution. Figure 2.2-2 shows that the distribution of X values is not strictly normal. Nevertheless the small observed values of the standard deviation of the sample XMEAN and the general smoothness of the curves indicate that the XMEAN curve in Figure 2.2-1 is a close approximation of the true XMEAN.

In Figure 2.2-1 data points are not given for values of N greater than 20, because the search exhausted the available storage. This occurred in the following way. The sample set of 895 problem instances was generated prior to the performance measurement experiments. The method of generating those instances (see Section 2.7)

¹⁵ For readers interested in the distribution of N , from 1 to 30, for a random problem instance, [Schofield 1967, p.131] reports the following information for the subset of problem instances in which the hole is in the center in both the initial board configuration and in the goal board configuration. Below, Q denotes the percentage of such problem instances (s_r, s_g) for which $h(s_r, s_g) = N$.

N	Q	N	Q
0	.005	18	12.2
4	.04	20	19.9
6	.04	22	25.8
8	.2	24	28.4
10	.4	26	9.4
12	1.2	28	1.8
14	2.8	30	.3
16	6.2		

¹⁶ Look closely, as the vertical range of the bars is small in Figure 2.2-1.

allowed us to determine the value of N for each instance. The instances were then grouped so that during the performance measurement experiments, searches were executed first for all instances having $N = 2$, followed by all instances having $N = 3$, and so. In the case of Figure 2.2-1 the searches for all instances having $N < 20$ terminated successfully; however, there was an instance having $N = 21$ which exhausted the available storage. At that point the experiment was terminated, and no results reported for any instances having $N \geq 20$. Hence Figure 2.2-1 represents 760 algorithm executions instead of the full 895 in the sample set. This explains also subsequent figures in which data for large N are omitted.

Figure 2.2-3 and 2.2-4 are analogous to Figure 2.2-1, showing XMEAN performance results when using K_2 and K_3 , respectively, instead of K_1 . Figure 2.2-5 plots the XMEAN data from Figures 2.2-1, 2.2-3, and 2.2-4.

Figure 2.2-6 shows the range of observed values of L for K_3 . Note that K_3 always finds minimal length solution paths (i.e., $L = 1$) for $N \leq 9$, with $LMEAN(K_3, .5, N) \leq 1.3$ for larger N . Note the decrease in $LMAX(K_3, .5, N)$ as N approaches 26. This observed decrease is due at least in part to the fact that the sample set includes fewer problem instances for $21 \leq N \leq 26$ than for $N < 21$. For human subjects solving instances of the 8-puzzle, $1.1 \leq L(s_p, s_g) \leq 3.3$ has been reported [Hayes et.al. 1965], [Doran & Michie 1966].

2.3 Parameter Tuning: Effects of Changing Term Weight

This section presents experimental results concerning how performance varies with the relative weight given a "forward looking" term and a "backward looking" term in the evaluation function $F(s)$. We achieve this by generalizing $F(s)$ from $F(s) = G(s) + K(s)$ (used in previous sections) to $F(s) = (1 - W) * G(s) + W * K(s)$, where W is defined on the interval $0 \leq W \leq 1$. To show how performance varies with W , we repeat the experiments of Section 2.2 for the cases $W = .1, .2, .3, .4, .6, .7, .8, .9$, and 1.0 . Note that the case $W = .5$ corresponds to $F(s) = G(s) + K(s)$, because it is the relative values and not the absolute values of $F(s)$ that determine the order in which

nodes are expanded. Similarly, $W = 0$ corresponds to $F(s) = G(s)$, yielding breadth-first search (reported in Section 2.2). Of particular interest is the comparison between $W = .5$ and $W = 1.0$. Certain formal analyses [Pohl 1970a], [Pohl 1970b], [Nilsson 1971], [Vanderbrug 1976] suggest the value of the $G(s)$ term for "insurance", but these results are restricted to K functions satisfying $KMIN(N) = N - e$ and $KMAX(N) = N + e$, and hence do not apply to the present 8-puzzle K functions, since the evidence given in Figures 2.4-2, 2.4-3, 2.4-4 indicate that the values of $KMIN(N)$ and $KMAX(N)$ do not grow linearly with N for the three K functions tested here. Some researchers [Pohl 1970b], [Vanderbrug 1976] prefer $W = 1.0$ to $W = .5$ for intuitive reasons. Here we provide experimental evidence; Section 3.4 provides analytic results.

Figure 2.3-1 shows how $XMEAN(K_1, W, N)$ varies with W , for $W = 0.2, 0.5, 0.7$, and 1.0 . Figures 2.3-2 and 2.3-3 show corresponding results for K_2 and K_3 . A separate execution of A^* was performed for each problem instance in the sample set, for each of these three K functions, for each value of W , for a total of $895 * 3 * 10 = 26850$ distinct executions.

Note in Figure 2.3-2 that for K_2 , as W increases the functional form of $XMEAN$ becomes subexponential in N , and that for fixed "medium-sized" values of N , $XMEAN(K_2, N, W)$ increases as W increases. Figure 2.3-4 displays this effect more prominently. That figure is plotted from the same data as is Figure 2.3-2, but includes points representing more distinct values of W (and fewer of N) than are displayed in Figure 2.3-2. The observation noted above merits explanation, because none of the reports cited in this chapter predict its occurrence, even on intuitive grounds. The dependence of $XMEAN(N)$ on W for K_1 (Figure 2.3-1) and for K_3 (Figure 2.3-3) is similar to that observed for K_2 (Figure 2.3-2), except that K_3 apparently "reaches its limit" at $W \sim .5$, i.e., larger values of W cause no significant change in $XMEAN$ performance. A rigorous explication of the latter observation remains for now an open problem. Figure 2.3-5 is analogous to Figure 2.2-5, but uses $W = 1.0$ instead of $W = .5$.

Together, we observe of the data plotted in Figures 2.3-1 through 2.3-5 that $W = 1.0$ minimizes the average number of nodes expanded if N is large with respect to the maximum value of N (i.e., with respect to $N_{max} = 30$ for the 8-puzzle), but for smaller N increasing W decreases cost up to a certain value of W , and increases it for larger W .

Note in Figure 2.2-5 that for almost every N , $XMEAN(K_3, .5, N) \leq XMEAN(K_2, .5, N) \leq XMEAN(K_1, .5, N)$. The "error bars" in that figure suggest that the exceptions may be statistically insignificant. (However, we have done no formal statistical analysis to determine whether this is strictly true.) The same is observed in Figure 2.3-5 for $W = 1.0$, and is observed generally among the data for each value of W measured. These observations suggest that under some conditions experimental results for problem instances of "small" distance N or of "medium" distance N accurately predict relative performance for problem instances of larger distance N . This may not be true generally. Figures 2.3-6, 2.3-7, and 2.3-8 are analogous to Figures 2.3-1, 2.3-2, and 2.3-3, but show $LMEAN(K, W, N)$ instead of $XMEAN(K, W, N)$.

Figure 2.3-9 is analogous to Figure 2.3-5, comparing the three K functions by length of the solution path ($LMEAN(N)$) for $W = 1.0$. Note in that figure that K_3 outperforms K_2 and K_1 by $LMEAN$, i.e.,

$LMEAN(K_3, 1.0, N) \leq LMEAN(K_2, 1.0, N) \leq LMEAN(K_1, 1.0, N)$ for all N ,
whereas for $W = .5$

$LMEAN(K_3, .5, N) \geq LMEAN(K_2, .5, N) = LMEAN(K_1, .5, N) = 1$ for all N .

(See Figure 2.2-6.) Hence the data in Figures 2.3-5 and 2.3-9 support the conjecture that for $W = 1.0$, faster heuristics find shorter solution paths, whereas Nilsson conjectured that solution quality (L) had to be sacrificed for speed ($XMEAN$) when selecting an inadmissible heuristic function over an admissible one. Here then is one condition under which this is not true.

Figure 2.3-10 shows for each K function how L varies with W in the aggregate, plotting for each combination of K and W the mean value of $LMEAN(K, W, N)$ over all N for which $LMEAN$ data were recorded.

The preceding figures constitute evidence that for large N , $XMEAN$ and $LMEAN$ are inversely related for fixed K as W varies, but are positively related for fixed large W as K varies. This says that if N is large, speed can be traded for solution quality by changing W (holding K fixed), but a tradeoff between speed and solution quality cannot apparently be effected by changing K (holding W fixed). Figures 2.3-11, 2.3-12, and 2.3-13 show this cost/quality tradeoff explicitly as W varies from 0.1 to 1.0, for the

cases $N = 15$, $N = 20$, and $N = 25$ respectively. In the "medium-sized" case $N = 15$, the trade-off is not advantageous: increasing W beyond a certain value brings longer solution paths at greater cost. For the larger values $N = 20$ and $N = 25$, the tradeoff is advantageous. Whether or not these phenomena generalize beyond the 8-puzzle remains an open question pending the results of future experiments for other problems.

The next data presented in this section differ from most of the preceding in that they are addressed to executing a comparison between experimental measurements and analytic predictions of performance. Whereas Section 2.4 presented similarly motivated data for the purpose of identifying the input to the model for a given comparison, here we provide the observed data against which the output of the model is to be compared. Formulas in Chapter 3 give values for the number of nodes expanded as a function of K , W , and N . Since Chapter 3 gives worst case predictions, we expect ipso facto those predictions to agree more closely with observed worst case performance ($XMAX(K, W, N)$) than with observed average case performance ($XMEAN(K, W, N)$). Figure 2.3-14 shows $XMAX(K_2, W, N)$ for all N represented in the sample set and for several values of W . Note the similarities in form between the curves in this figure and the corresponding curves in Figure 2.3-2.

The $XMEAN$ data presented in this section can be interpreted to provide evidence concerning Nilsson's conjecture about the efficacy of K functions that overestimate h (see Section 2.2). This follows from the observation that changing the value of W is equivalent to not changing W , but instead multiplying a fixed K function by a particular scalar value while holding W fixed. This holds because it is the relative values and not the absolute values of $F(s)$ that determine the order in which nodes are expanded. Hence the functions $F(s) = (1 - W) G(s) + W K(s)$ and $F(s) = G(s) + V K(s)$ have equivalent effect if the ratios of the weights given to the $G(s)$ term and to the $K(s)$ term are identical, i.e., if $V / 1 = W / (1 - W)$. Hence the combination (W, K) is equivalent to $(.5, V K) = (.5, K W / (1 - W))$. The case $v > 1$ corresponds to $W > .5$, and for example, $W = .7$ corresponds to $V = 2.33$. Under this interpretation, the data in Figures 2.3-1 through 2.3-5 for $W > .5$ support Nilsson's expectations so long as N is large; as noted earlier in this section, the expected decrease in $XMEAN(W, N)$ for fixed "mid-sized" N as W increases fails to materialize; the cost in fact increases as W increases for this case.

2.4 Cost vs. Error in Heuristic Estimates of Distance to the Goal

This section reports measurements of the range in each of the three K function's estimates of distance to the goal, as a function of the actual distance to the goal. These data differ somewhat from those reported in other sections of this chapter, in character and in purpose. They differ in character in measuring the values computed by a K function, as opposed to values of the resulting search performance. In purpose, these data provide a means to map a given K function occurring in practice to corresponding functions defined in the simplified model defined in Chapter 3. In Chapter 3 we derive a general formula (in fact, several) for the performance associated with an arbitrary K function, as a function of the distance-estimate values it computes. The data reported in this section serve as actual parameters to this formula.

The model proposed by [Pohl 1970a] gives a formula for the number of nodes expanded, valid for any K function satisfying the following condition for all s:

$$h(s) - e \leq K(s) \leq h(s) + e \quad (2.4-1)$$

Here we generalize (2.4-1) to the form:

$$KMIN(h(s)) \leq K(s) \leq KMAX(h(s))$$

or equivalently:

$$KMIN(N) \leq K(s) \leq KMAX(N), \text{ where } N = h(s) \quad (2.4-2)$$

Hence we generalize Pohl's definition of the bounding functions of a K function to permit arbitrary KMIN(N) and KMAX(N) functions, so that corresponding to any given K function for a problem graph Q there exist particular KMIN(N) and KMAX(N) functions. Formally:

Definition: Bounds on heuristic distance estimates

KMIN(Q, K, N) is defined to be the minimum value of $K(s_i, s_j)$ over all node pairs (s_i, s_j) in $U(Q)$ such that $h(s_i, s_j) = N$.

Functions KMAX(Q, K, N) and KMEAN(Q, K, N) are defined similarly.

Measurements of the distance-estimate values computed by K_3 were reported in [Doran & Michie 1966, p. 248]. Specifically, they report the values of $K_3(s_i)$ for the 18 nodes s_i along the solution path found for the problem instance cited in Section 2.2. To the best of our knowledge, no other measurements of $KMIN(Q, K, N)$ or $KMAX(Q, K, N)$ or $KMEAN(Q, K, N)$ data for any Q , K , and N have appeared to date in the literature.

In principle, given any Q and K the exact values of $KMIN(Q, K, N)$ and $KMAX(Q, K, N)$ are straightforward to determine by exhaustive enumeration. In the case of the 8-puzzle, however, there are about $6 \cdot 10^{10}$ pairs (s_i, s_j) , making an exhaustive enumeration computationally infeasible.

Instead, we use a sample of the (s_i, s_j) node pairs to determine an approximation to the exact $KMIN$ and $KMAX$ values for heuristic functions K_1 , K_2 , and K_3 , as follows. For each of the 895 problem instances (s_r, s_g) in the sample set defined in Section 2.2, we know the value of $h(s_r, s_g)$ (from the manner in which the instances were generated; see Section 2.7), and we compute the value of $K_1(s_r, s_g)$. For each of the 895 problem instances, we set the value of $KMIN(K_1, h(s_r, s_g))$ to be the smaller of $K_1(s_r, s_g)$ and the current value of $KMIN(K_1, h(s_r, s_g))$ (which value had been initialized to a very large value). Similarly, we set the value of $KMAX(K_1, h(s_r, s_g))$ to be the larger of $K_1(s_r, s_g)$ and the current value of $KMAX(K_1, h(s_r, s_g))$ (which value had been initialized to a very large negative value). In this way we determine an approximation to $KMIN(K_1, N)$ and $KMAX(K_1, N)$ based on from 8 to 40 samples per value of N . This approximation, of course, determines a lower bound on the values of $KMAX(K_1, N)$ and an upper bound on the values of $KMIN(K_1, N)$. We repeat this procedure for K_2 and K_3 in turn in place of K_1 .

Clearly, the values obtained by this approximation depend on the number of samples on which the approximation is based. Furthermore, we know from the results of order statistics (e.g., [Barlow 1972], [David 1970], [de Haan 1976], [Gumbel 1958]) that the observed maximum (minimum) of a random sample taken over a finite distribution is a biased estimator of the exact maximum (minimum) of the distribution (because the maximum of the distribution is as large or larger than the sample maximum). The results of order statistics, however, are problematic to apply in the present case because we are generally uninformed about the distribution of the values of $K(s_i, s_j)$ over the set of all (s_i, s_j) in the 8-puzzle, where K denotes K_1 , K_2 , or K_3 .

(See Figure 2.2-2.)

As an alternative to attempting to apply order statistics techniques, we measure directly the dependence of our estimates of KMIN and KMAX values on the number of samples. To do so we repeat the approximation of KMIN and KMAX values described above (call it sample A) using a larger sample of (s_i, s_j) for which both $h(s_i, s_j)$ and $K(s_i, s_j)$ can be determined. This second sample (sample B) consists of the 895 problem instances (s_r, s_g) in the sample set of Section 2.2, and, for each such pair, the nodes along the solution path between them found using K_2 and $W = .5$. The A^* admissibility theorem [Hart et.al. 1968] insures that any solution path found using K_1 or K_2 with $W \leq .5$ is of minimal length. Hence we can determine the value of $h(s_i, s_g)$ for nodes s_i along the solution path found from s_r to s_g : if s_i denotes the node on the solution path that is i steps away from the goal node s_g , then $h(s_i, s_g) = i$ by the admissibility theorem. For each problem instance of distance N we compute $K(s_i)$ and update $KMAX(i)$ and $KMIN(i)$ for each $1 \leq i \leq N$. Hence for each of the three K functions, each problem instance of distance N in the sample set contributes N distinct observations of the values of $K(s_i, s_g)$ vs. $h(s_i, s_g)$ to the KMIN and KMAX estimates. Hence sample B includes 11,448 (s_i, s_j) observations, about 10^{-7} of the total population in the 8-puzzle. Note however that these observations are not distributed uniformly with respect to N . If $t(N)$ denotes the number of observations upon which the experimental estimates of $KMIN(N)$ and $KMAX(N)$ are based, and if $u(N)$ denotes the number of problem instances of distance N in the sample set, then $t(N) = \sum_{N \leq i \leq 26} u(i)$. Hence $t(26) = u(26) = 8$, $t(25) = u(25) + u(26) = 20, \dots, t(3) = 855$, $t(2) = 895$.

Table 2.4-1 compares the KMIN and KMAX estimates obtained using sample A with those obtained using sample B, for both K_1 and K_2 . For the reader's convenience, a "*" identifies those entries for which the two estimates of $KMIN(i)$, or $KMAX(i)$, disagree.

i	K_1		K_2		K_2		K_2		No. of samples	
	KMIN(i) A	B	KMAX(i) A	B	KMIN(i) A	B	KMAX(i) A	B	A	B
0		0		0		0		0		895
1		1		1		1		1		895
2	2	2	2	2	2	2	2	2	40	895
3	3	3	3	3	3	3	3	3	40	855
4	3	3	4	4	4	4	4	4	40	815
5	3	3	5	5	5	5	5	5	40	775
6	3	3	6	6	4	4	6	6	40	735
7	4	4	7	7	5	5	7	7	40	695
8	3	3	8	8	4	4	8	8	40	655
9	3	3	8	8	5	5	9	9	40	615
10	2	2	8	8	4	4	10	10	40	575
11	5	1 *	8	8	7	3 *	11	11	40	535
12	4	2 *	8	8	6	4 *	12	12	40	495
13	3	3	8	8	5	5	13	13	40	455
14	4	3 *	8	8	6	4 *	14	14	40	415
15	4	3 *	8	8	5	5	15	15	40	375
16	3	3	8	8	4	4	16	16	40	335
17	4	4	8	8	5	5	15	17 *	40	295
18	4	4	8	8	4	4	18	18	40	255
19	5	5	8	8	7	7	19	19	40	215
20	4	4	8	8	4	4	20	20	40	175
21		4	8	8	7	7	19	21 *	30	135
22		5	8	8	12	10 *	20	20	30	105
23		5	8	8	9	9	21	21	30	75
24		5	8	8	12	10 *	22	22	25	45
25		6	8	8	11	11	19	21 *	12	20
26		6	8	8	10	10	20	20	8	8

Table 2.4-1 Comparison of KMIN(i) and KMAX(i) values for K_1 and K_2 using different-sized samples

Table 2.4-1 indicates that samples A and B, although differing by more than an order or magnitude in cardinality, yield identical estimates of KMIN and KMAX values in all but a few cases. We take this as evidence that the KMIN and KMAX approximations based on sample B are close to the exact values.

Figure 2.4-1 plots the KMIN(N) and KMAX(N) estimates obtained for K_1 using sample B. Figures 2.4-2 and 2.4-3 plot the analogous results for K_2 and K_3 , respectively. Comparison of Figures 2.4-1, 2.4-2, and 2.4-3 with condition (2.4-1) serves to motivate the generalization of Pohl's model that is reported in Chapter 3:

Pohl's assumptions exclude the 8-puzzle K functions studied here. By deriving results general enough to include within their scope these K functions that occur in practice, the model of Chapter 3 is testable: the restrictiveness of its other assumptions can be assessed objectively by comparing model predictions with experimental observations. In Section 3.5 we do so using the 8-puzzle data reported in this chapter.

2.5 "Internal" Measures of Search Behavior

If pressed at this point to decide which two of K_1 , K_2 , and K_3 are more similar and which is relatively different from the other two, the quantitative data given in the preceding sections of this chapter suggest that the performance of K_3 differs qualitatively from those of K_1 and K_2 . In this section we attempt, for K_1 , K_2 , and K_3 , to relate differences in external search performance (XMEAN and LMEAN) with differences in search behavior as observed during search of an individual problem instance. We define two measures of behavior during search:

Definition: Search Behavior Measures

$LEV(Q, K, W, s_r, s_g, i)$ denotes the number of nodes occurring at level i in the search tree as it exists when A^* terminates. (Hence these nodes were found during the search at distance i from the root node.)

$RUN(Q, K, W, s_r, s_g)$ denotes the mean "run length" of the search, i.e., the number of nodes expanded, divided by one plus the number of "hops" that occur when the next node expanded is not a son of the last node expanded.

$LEVMEAN(Q, K, W, N, i)$ and $RUNMEAN(Q, K, W, N)$ are defined in the standard way.

For example, in the hypothetical search tree depicted in Figure 2.5-1 the runs between hops are (1, 2), (3, 4) and (5, 6, 7, 8), so $RUN = (2 + 2 + 4) / 3 = 2.66$. For breadth-first search $RUN = (X + 1) / X$, where X denotes the number of nodes expanded. For optimal search (only nodes along a single minimal length solution path are expanded), $RUN = X = N$, the depth of the goal.

Figure 2.5-2 shows $LEVMEAN(K_j, W, N, i)$ for $j = 1, 2, 3$, and the case $N = 20$ and

$W = .5$. Note that the maximum value of $LEVMEAN(i)$ occurs at about $i = N/2 = 10$ for each K_j , and that $LEVMEAN(i)$ is approximately symmetric about this value of i . (The values of $LEVMEAN(K_3, .5, 20, i)$ for $i > 20$ are not plotted in Figure 2.5-2 because they are less than 1.) Observe for K_1 and K_2 that $LEVMEAN(i)$ increases (and then decreases) approximately "exponentially" with i .¹⁷ In contrast, $LEVMEAN(i)$ for K_3 is distributed more uniformly with i , suggesting a possible relation between the growth rate of $XMEAN(N)$ for fixed K and the distribution of $LEVMEAN(N, i)$ with i for the same K (but this we leave for future investigation).

We can attempt to quantify the amount of "mid-depth bulge" exhibited by $LEVMEAN(i)$, as follows. For K_1 , the sum of $LEVMEAN(i)$ for $i = 11, 12$, and 13 is half the sum of $LEVMEAN(i)$ for all i . (Note that the latter value equals the value of $XMEAN$.) We say then that the 50% LEVMEAN-interval for K_1 is $[11, 13]$. Similarly, the 90% LEVMEAN-interval for K_1 is $[8, 15]$. Since $[1, 20]$ is the entire interval (i.e., $N = 20$), we define the 50% LEVMEAN-interval fraction to be $(13 - 11 + 1) / (20 - 1 + 1) = .15$. Similarly, the 90% LEVMEAN-interval fraction is $(15 - 8 + 1) / 20 = .4$. In general let $IF(p)$ denote the value of the p LEVMEAN-interval fraction. If $LEVMEAN(i)$ were uniformly distributed with i then $IF(p) = p$. Figure 2.5-3 plots $IF(p)$ vs. p for $p = .25, .5, .75$, and $.9$, for K_1, K_2 , and K_3 . Mid-depth bulge may be measured by $IF(p) - p$, by which K_1 and K_2 are easily distinguished visually from K_3 . A possible scalar measure of mid-depth bulge (MDB) is the simple arithmetic mean of $|IF(p) - p|$ over the available values of p and $IF(p)$. For the four values plotted for each K function in Figure 2.5-3 we have

$$MDB(K_1) = (.25 - .05 + .5 - .15 + .75 - .25 + .9 - .4) / 4 = .39$$

$$MDB(K_2) = .29$$

$$MDB(K_3) = .06$$

¹⁷ The terms "exponential" and "subexponential" are used here only for brevity's sake in indicating that the data plotted on a semilog scale appear to be closely approximated by a straight line, or appear to be closely approximated by a sublinear function, respectively. When we subsequently suggest that a certain "exponential" curve becomes "subexponential" as the value of a certain parameter varies, we mean only that the plotted data provide visual demonstration of a qualitative difference between two or more curves being compared.

These values suggest a possible relation between MDB for a given value of N and the growth rate of $XMEAN$ as a function of N , but we leave such investigations for future work (see E2-4 in Appendix B). Analogous $LEVMEAN(K, W, N, i)$ data were collected for all other tested combinations of K , W , and N , but these have not yet been analyzed, for reasons of sheer volume of data.

Figure 2.5-4 shows $RUNMEAN(K_i, N)$ for $i = 1, 2, 3$. $RUNMEAN(N) = N$ for small N because these K functions are optimal or nearly so for small N . Whereas the MDB measure distinguishes K_3 from K_1 and K_2 fairly sharply, by $RUNMEAN$ the difference is only of degree, suggesting no credible means of distinguishing subexponential from exponential cost heuristics by this measure. These data provide specific challenges: is the similarity in form of the three curves coincidental? And what is the significance of this particular three phase "decay" form, in which $RUNMEAN(N)$ increases approximately with N up to a point, followed by a sharp decrease with increasing N , followed by a less sharp decrease?

The fact that $RUNMEAN$ for K_1 is little more than 1 for large N , together with the mid-depth bulge data above, suggests that ordered depth-first search, using the same K function, may actually expand fewer nodes than A^* best-first search for large N using "poor" heuristic functions. This possibility is described further in E2-3 in Appendix B. As with $LEMEAN$, additional $RUNMEAN$ data were collected but has not yet been analyzed (see E2-4 in Appendix B).

2.6 Predictions about Performance of a Complex Best-First Search System

This section presents no experimental or analytic results. Rather, its objective is to suggest, by drawing a concrete analogy between simple search and complex search, that the results of experiments measuring the performance of complex best-first search systems might be worth the trouble of obtaining them. The program construction phase of the PSI program synthesis system [Barstow & Kant 1976], [Barstow 1977], [Green 1977] employs a search mechanism that could be implemented as a best-first search, but actually has been implemented as a branch and bound

algorithm [Kant 1977]. We interpret the current experimental results for the 8-puzzle as if they applied to some extent to PSI.

The program construction subsystem of PSI converts a high level program specification into a legal LISP implementation by applying rules that refine a specification into a slightly more detailed specification, and ultimately into primitives corresponding to segments of LISP code [Barstow 1977]. The coding rule set of PSI induces a tree in which the terminal nodes correspond to legal target programs for the given input specification. These target programs can differ drastically in efficiency, so that some goodness value may be assigned to each terminal node, comparable to the L measure here. Search of the entire tree to find the most efficient implementation (target program with smallest L) may incur prohibitive expense (i.e., X = number of nodes expanded in refinement tree). Hence an "efficiency expert" (comparable to a K function, albeit a rather complex one) guides the search in an attempt to keep both L and X acceptably small [Barstow & Kant 1976], [Kant 1977]. The variable N, the number of refinement steps, might refer to the length of the shortest path (number of successive steps) in the refinement tree to the terminal node (target program) found by the search, or alternatively to the length of the path to the terminal node representing the most efficient target program of all possible legal programs.

The 8-puzzle experimental results support the following conjectures about the performance of this phase of PSI, assuming best-first search with an evaluation function comparable to $F(s) = (1 - W) * G(s) + W * K(s)$.

- 1) For $W = .5$, unless the efficiency expert estimates what corresponds to distance "very accurately", it will not be feasible to synthesize target programs that require very many refinement steps (i.e. large N), because the number of nodes expanded (i.e., $XMEAN(N)$) will grow exponentially with N. (See Figure 2.2-5)
- 2) Simply by choosing $W = 1.0$ instead of $W = .5$, $XMEAN(N)$ becomes sub-exponential: It will cost somewhat more to synthesize "medium-sized" target programs than if $W = .5$, but far less to synthesize "large" target programs (Figures 2.3-1 through 2.3-5). However, the synthesized target programs may be less efficient than if $W = .5$ is used (Figures 2.2-6, 2.3-9).
- 3) If the efficiency expert is improved so as to reduce $XMEAN(N)$, then the improvement will be observed for every value of W (Figures 2.2-5, 2.3-5; compare K_1 to K_2 , or K_2 to K_3 or K_1 to K_3). Furthermore, for large W, the improvement in speed will also cause an improvement in the efficiency of the synthesized target

programs: if heuristic K_2 causes target programs to be synthesized more quickly than does K_1 , then also those target programs synthesized by K_2 are computationally more efficient to execute than those synthesized by K_1 (Figure 2.3-5, 2.3-9).

- 4) The XMEAN performance of a version of the efficiency expert for "large" N can be predicted by measuring mid-depth bulge for "medium-sized" N (Figures 2.5-2, 2.5-3).
- 5) The target program will hop around the search tree quite a lot unless N is small (Figure 2.5-4). Mean run length < 2 indicates "poor" efficiency expert. In this case, ordered depth-first search may prove to be more efficient than best-first search.

The extent to which the above predictions accurately reflect performance in the hypothetical PSI-like system described above can be determined straightforwardly by experiment with such a system.

2.7 Procedure for Generating Random Problem Instances

This section describes the method used to generate the sample set of 895 problem instances of the 8-puzzle used in the experiments described in Sections 2.2, 2.3, 2.4, and 2.5. We define a one-to-one, onto mapping (i.e., bijection) to exist from the $9!$ permutations of the sequence $0,1,2,\dots,8$ to the $9!$ nodes (representing distinct board configurations) of the 8-puzzle graph. Given a permutation $\pi = \pi(1),\pi(2),\dots,\pi(9)$, its image under the bijection is constructed to obtain a random board configuration, represented as a vector $B[1:9]$, where $B[i] = \pi(i)$ denotes that the i 'th board square (numbered in left to right, top to bottom order) is assigned the tile numbered $\pi(i)$, except that $\pi(i) = 0$ denotes that the square is empty (i.e., is occupied by the hole). A pseudo-random number generator is used to select permutations such that each is equally likely to be selected.

For each initial node s_r generated using the above procedure, we generate a goal node s_g , and hence a problem instance (s_r, s_g) . We generate first a set of problem instances for which $N = h(s_r, s_g) = 1$, then independently a similar set for $N = 2$, and so on until $N = 26$. For each value of N in this range, the sample set is produced as follows.

Generate an initial configuration s_0 as above. Applying the operators MOVE-HOLE-RIGHT, MOVE-HOLE-LEFT, MOVE-HOLE-DOWN, and MOVE-HOLE-UP to s_0 , if applicable, generate the set of all board configurations that differ from s_0 by a single legal tile move. Choose one of the latter randomly, such that each is equally likely. Call the chosen configuration s_1 . Similarly, obtain the successors of s_1 (excluding s_0) and select one randomly, calling it s_2 . Proceed in this way until a sequence $s_0, s_1, \dots, s_N$ is obtained. Note that this method insures that s_0 and s_N are members of the same component of the 8-puzzle graph. The pair (s_0, s_N) is included in the sample set if and only if $h(s_0, s_N) = N$ (i.e., if the generated path from s_0 to s_N is of minimal length).

The latter condition is tested by executing A^* on problem instance (s_0, s_N) using heuristic function K_2 with $W = .5$. As noted previously, A^* finds minimal length solution paths when executed with these actual parameters, so if the solution path found is of length less than N , then this instance is not included in the sample set. This process is repeated for each (s_0, s_N) generated as above for a given value of N , until 40 instances (for $N \leq 20$) have passed this filter. The percentage of instances that are rejected is observed to be small for small N , but grows quite large as N increases. Hence for practical reasons we generated smaller numbers of samples for $21 \leq N \leq 26$: 30 samples for each of $N = 21, 22$, and 23 ; 25 samples for $N = 24$; 12 samples for $N = 25$; and 8 samples for $N = 26$.

Note that the filter is a filter when generating problem instances, as described in this section. It acted from and preceded the 26,850 A^* executions made for the purpose of measuring performance quantities.

Formally, let

$$T_Q(N) = \{ (s_r, s_g) \mid (s_r, s_g) \in U(Q) \text{ and } h(s_r, s_g) = N \}$$

We believed originally that the instances in the sample set are chosen independently and uniformly from $T_Q(N)$. Subsequently, however, it was pointed out that this is not necessarily the case, for the following reason. "To the contrary, if $m(s_0, s_N)$ is the number of paths of length N (the minimal length) connecting s_0 to s_N , I believe that the probability that (s_0, s_N) will appear in your sample is proportional to $m(s_0, s_N)$, which is not in general a constant. In fact $m(s_0, s_N)$ can be taken as a measure of difficulty of the problem. With your sample biased toward having 'easier'

problems, you favor algorithms tending toward depth rather than breadth. Consequently this biased sample could easily affect the results." [Kadane 1978]

To determine whether the sample described above does indeed bias the results, we generated a new sample of problem instances and repeated some of the experiments. A problem instance is generated in the new sample set by selecting randomly (i.e., uniformly, independently, and with replacement) two permutations of the integers 0 through 8. Each such permutation corresponds to a tile configuration of the 8-puzzle. An arbitrary pair of such tile configurations is not necessarily a solvable problem instance however, because the two configurations may not necessarily belong to the same component of the 8-puzzle graph. Schofield [1967] however describes a necessary and sufficient test for deciding this solvability question for the 8-puzzle. We adapt his procedure to the current task; hence we generate a sequence of pairs of tile configurations and discard those that are not solvable. As mentioned earlier, problem instances of the 8-puzzle are not distributed uniformly with respect to the value of N . Accordingly, we generated problem instances randomly in this new sample set, but retained only as many as 60 instances for any value of N . Excluding values of N for which only a small number of samples were obtained before terminating the generation process, the resulting sample set numbered 445 problem instances. Using this sample set, we repeated the experiments reported in Figures 2.2-1, 2.2-3, and 2.2-4. Table 2.7-1 compares the results based on the first sample set (called set 1 below) with those based on the new sample set (set 2), for common values of N . The columns in Table 2.7-1 labeled "sdsm" give the sample standard deviation of the sample mean, a measure of how closely the observed value of XMEAN approximates the true value of XMEAN. Cases in which the difference between the XMEAN value of set 1 and that of set 2 exceeds the sum of the corresponding sdsm value of set 1 and that of set 2 are indicated by a "*".

K1:				SET 2			
SET 1			no. of samples	SET 2			no. of samples
N	XMEAN	sds		XMEAN	sds		
16	426.4	15.6	40	439.0	19.0	16	
17	746.7	21.0	40	700.0	17.0	33	
18	1061.7	26.8	40	1130.2	32.8	41	
19	1767.9	33.6	40	1751.4	36.4	58	
20	2659.4	57.7	40	2736.1	55.4	60	

K2:				SET 2			
SET 1			no. of samples	SET 2			no. of samples
N	XMEAN	sds		XMEAN	sds		
16	79.4	9.6	40	72.7	9.7	16	
17	119.2	12.2	40	128.0	11.9	33	
18	161.9	16.1	40	176.3	16.4	41	
19	237.4	22.3	40	210.4	14.6	58	
20	267.0	24.3	40	313.9	22.2	60	
21	351.6	36.1	30	401.0	25.2	60	
22	422.7	47.1	30	635.1	40.3	60	
23	691.5	74.0	30	711.8	59.8	60	
24	1042.9	115.1	25	1127.9	75.5	60	
25	1552.4	288.3	12	1431.0	84.5	60	
26	1958.5	541.9	8	1998.5	120.2	57	

K3:				SET 2			
SET 1			no. of samples	SET 2			no. of samples
N	XMEAN	sds		XMEAN	sds		
16	32.2	4.0	40	57.2	10.7	16	
17	55.4	5.9	40	66.7	9.3	33	
18	53.9	6.4	40	66.8	8.2	41	
19	66.3	8.1	40	72.5	5.8	58	
20	62.1	7.3	40	71.8	4.9	60	
21	59.9	6.5	30	95.1	7.6	60	
22	92.5	11.0	30	100.38	9.1	60	
23	93.9	16.3	30	103.2	8.6	60	
24	81.2	9.5	25	96.3	7.0	60	
25	99.9	28.8	12	101.4	10.4	60	
26	57.3	7.7	8	115.2	12.4	57	

Table 2.7-1 Comparison of two sample sets of problem instances

The data tabulated above indicate that estimates based on these two sample sets differ only by a small amount in most cases. Without using much larger sample sets, it is difficult to determine what fraction of the observed differences between the

two sets is due to bias in sample set 1, and what fraction simply to variance (i.e., to differences between the observed estimate and the actual value for each case within each sample set). Clearly, differences such as those tabulated above merit explanation in future investigations of this sort.

2.8 Statistical Issues

The experimental data reported in this chapter are estimates of the values of certain well-defined mathematical functions evaluated at particular argument values. This section concerns the statistical question of determining the differences between the estimates and the actual values to which they correspond. The estimates in question fall into two categories: estimates of mean values (i.e., estimates of values of the XMEAN, RUNMEAN, and LEVMEAN functions), and estimates of extrema (i.e., estimates of the values of the KMIN and KMAX functions). We address questions of statistics for these two categories in turn, and consider as well what significance answers to these statistical questions might have to progress in artificial intelligence research.

Section 2.4 reports experimental estimates of $KMIN(K, i)$ and $KMAX(K, i)$ for three K functions and a range of values of i . That section also discusses the difficulties in applying the mathematical results of order statistics to the present application, and reports the results of a direct experiment to determine the dependence of the estimates obtained on the number of samples taken. This experiment gave evidence that a relatively few samples suffice to give accurate estimates of KMIN and KMAX values: extending the cardinality of the sample set by a factor of ten gave identical estimates in all but a few cases. This sensitivity experiment is subject to criticism on purely statistical grounds, in that the second larger sample is not drawn independently from the same universe as in the first sample set. This deficiency is mitigated by two factors: (1) additional samples increase monotonically the accuracy of a sample extremum (which is not necessarily the case for a sample mean); and (2) the values in the second sample set could be obtained at little additional computational expense

beyond that required for the first sample set, whereas applying the methodology used in obtaining the first sample to a much larger sample is computationally infeasible. Hence we obtained what additional evidence seemed possible.

One can argue that determining precise statistical bounds on the accuracy of experimental data in AI research is relatively less important than obtaining any experimental results at all. Inspection of the AI literature bears out this viewpoint historically: If experimental data are presented at all, they usually represent either individual cases or ensembles for which only mean value statistics are presented. Historical precedent of course does not justify an inadequate methodology; on the contrary, it simply reflects the practical necessity, in exploring a relatively new domain, for trading precise but few results for a larger volume of results that may raise specific questions appropriate for subsequent more detailed analysis. Chapter 1, especially Section 1.5, discusses this issue of trade-offs in the present exploratory investigation.

The fact that the present estimates of KMIN and KMAX values are used subsequently in Section 3.5 as the basis for testing the accuracy of analytic predictions, however, makes the issue of statistical precision more important than if the results were obtained simply as ends in themselves. The KMIN and KMAX values reported in Section 2.4 constitute some of the argument values at which a formula for the XWORST function is evaluated to test the predictive accuracy of that formula. Since the predictions vary with the input values on which they are based, one can attempt to account for the observed discrepancies between analytic prediction and experimental observation by attributing the discrepancies individually to different factors, including the accuracy of the KMIN and KMAX values. Hence the evidence cited in Section 2.4 -- suggesting that the observed KMIN and KMAX values are relatively accurate estimates of the actual extrema values to which they correspond -- is addressed precisely to this point. A detailed accounting of the discrepancy between prediction and observation is beyond the scope of this dissertation, one of whose more modest objectives was simply to demonstrate the technical feasibility of making any predictions whatsoever about A^* cost for particular realistic cases. Hence the present evidence serves simply to indicate the type of approach a subsequent more detailed "discrepancy analysis" might take.

We turn now from estimates of extrema to estimates of mean values. As discussed in Section 2.2, many of the figures in this chapter plot the sample standard deviation of the sample mean (sds_m) -- a measure of confidence in the accuracy of the sample mean. This measure can be unreliable, however, if the number of samples is small and the samples include outliers -- values differing from the actual mean by a large factor. Hence in addition simply to reporting sds_m values, in several cases we reported additional information about the sample distribution. Specifically, Figures 2.2-1, 2.2-3, 2.2-4, and 2.2-6 plot sample maximum and minimum values as well as sample mean values, as a function of the parameter N . For the case of Figure 2.2-3, we plotted in Figure 2.2-2 the frequency distribution of observed values for selected values of N .

To supplement these data, we now provide additional data showing directly the dependence of the sample mean on the number of samples. The data concern the estimates of $XMEAN(K, .5, N)$ plotted in Figures 2.2-1, 2.2-3, and 2.2-4 for heuristic functions K_1 , K_2 , and K_3 , respectively, for a range of values of N . (These three curves are plotted together in Figure 2.2-5.) Table 2.8-1 lists the estimates of $XMEAN(K, .5, N)$ obtained using 10, 20, 30 and 40 samples, for each combination of the three K functions and $N = 10, 15$, and 20. To these we add analogous data for 10, 20, and 30 samples for the case $N = 23$. In this table, D denotes the difference between the estimates derived from 30 and 40 samples, divided by the estimate derived from 40 samples (except that for $N = 23$, D denotes the fractional difference between the estimates derived from 20 and 30 samples).

K ₁ :					
no. of samples:	10	20	30	40	0
N: 10	29.9	31.9	30.9	29.1	6.1%
15	255.0	265.8	271.1	271.6	0.2%
20	2524.8	2578.0	2646.8	2659.4	0.5%

K ₂ :					
no. of samples:	10	20	30	40	0
N: 10	17.1	16.0	15.6	14.4	8.9%
15	41.3	49.1	55.2	55.8	1.1%
20	190.1	199.8	245.1	267.1	8.2%
23	720.0	661.4	691.5		4.4%

K ₃ :					
no. of samples:	10	20	30	40	0
N: 10	14.6	13.2	14.4	13.65	5.5%
15	35.7	38.1	39.3	36.7	7.2%
20	61.7	65.9	63.7	62.1	2.6%
23	112.8	102.1	93.9		8.7%

Table 2.8-1 Comparison of XMEAN based on different numbers of samples

These data show that the inclusion of the last ten samples in the sample sets had the effect of changing the estimate by less than 9% in each case.

Together, the evidence cited and reported above suggests that the effect of outliers on estimates of XMEAN is not excessive. Of course, there are other estimators which are less sensitive to the existence of outliers, such as the median or the arithmetic mean of the first and third quartiles. It would be useful to compare such statistics to the sample mean in future experiments of the sort described in this chapter.

This section has described the pragmatic role for statistical techniques used in the present work. Although relatively more attention has been devoted to such issues in this work than in many other reports of AI research, the discussion makes it clear that statistical techniques can be exploited to a much greater extent than at present in determining the accuracy of experimental data derived from Monte Carlo experiments.

2.9 Conclusions and Future Experiment

It has been opined that "The problem of efficiently searching a graph has essentially been solved and thus no longer occupies AI researchers" [Nilsson 1974, p. 787]. The experimental results reported in this chapter, and the further questions they raise, suggest that this statement is premature.

The present data confirm, qualify, or contradict various previous conjectures or generalizations about A^* performance appearing in the literature. Furthermore, the plotted data also provide visual demonstration of the existence of certain intriguing and previously unsuspected patterns and regularities in the performance data. These specific results are enumerated in Section 2.0. Below we comment on several instances of new phenomena revealed by the present data, and on several methodological issues.

The heuristic function K_3 apparently reaches a limit in improvement of $XMEAN(N)$ performance as W increases, whereas K_1 and K_2 apparently do not. Can this be attributed in entirety to the fact that K_3 overestimates h whereas K_1 and K_2 do not? In either case, the data constitute evidence of the existence of qualitative differences among K functions, raising the need to define sharp criteria by which to distinguish disjoint categories of K functions.

As a second example of previously unsuspected patterns revealed by these experiments, the data show a decrease in $XMEAN(N)$ for N large with respect to the diameter of the graph as W increases (as suggested by speculation in the literature), but the same data show a concomitant increase in $XMEAN(N)$ for "mid-sized" N as W increases beyond a certain value. How general is this phenomena, i.e., to what factors can it be attributed?

The data are voluminous: the number of data points reported here is several orders of magnitude more than in previous experiments with the 8-puzzle. The volume of data indicates the practical usefulness of a data base containing the experimental data, and of a data analysis program to operate thereon.

Note that the data become relatively more interesting in Section 2.3, with the introduction of variations in the weighting parameter W . But the cost of doing

experiments increases correspondingly, since the executions over the sample set must be repeated for each value of W selected. This makes more apparent the desirability of mathematical or simulation models to answer the same questions. Chapter 3 presents an analysis of one such model.

As an example of experimental mathematics research using computers, the present work contrasts with that of [Wells 1965], who by exhaustive enumeration determined that no algorithm could sort 12 items using fewer than 30 comparisons.¹⁸ The exhaustive enumeration approach is not computationally feasible in the present case, because the cardinality of the set consisting of all problem instances of the 8-puzzle graph is approximately 10^{10} . On the other hand, in the present work the mean values of functions over their domains are of greater interest than are maximum or minimum values, hence the Monte Carlo method as used here is both practical and methodologically acceptable. We suggest that the results reported here constitute an instance of "experimental analysis of algorithms" research, in the sense in which the term might be used to describe the results cited by [Weide 1977, p.303]. It would be interesting to derive general formulas for $XMEAN(Q, K, W, N)$ and for $XMAX(Q, K, W, N)$, but the generality of the model makes this difficult. Chapter 3 derives general formulas for a related but simpler model.

Clearly much remains to be investigated about the performance of the A^* algorithm under various conditions. Possibilities for future experiments include the following:

- 1) performing experiments for other problems analogous to those reported in this chapter; (This is discussed in more detail under item E2-1 in Appendix C)
- 2) attempting to exploit our discoveries about the dependence of cost on W , by varying W dynamically during a search in such a way as to minimize the cost of the search (item E2-2 in Appendix C);
- 3) following up our discoveries about the number of "hops" A^* makes in the search tree (i.e., as illustrated by the RUNMEAN data), to determine whether ordered depth-first search outperforms A^* when using heuristic functions that cause a lot of "hops" in A^* (item E2-3 in Appendix C);

¹⁸ Wells determined the minimum value of a particular function over the elements of its domain. See [Weide 1977, p.303] for discussion of this and other examples.

- 4) relating the dependence of $XMEAN(N)$ on W to corresponding changes in $LEVMEAN(N)$ and $RUNMEAN(N)$ with W (item E2-4 in Appendix C);
- 5) comparing the performances of different heuristic functions for individual problem instances instead of comparing average performance over an aggregate of problem instances (item E2-5 in Appendix C).

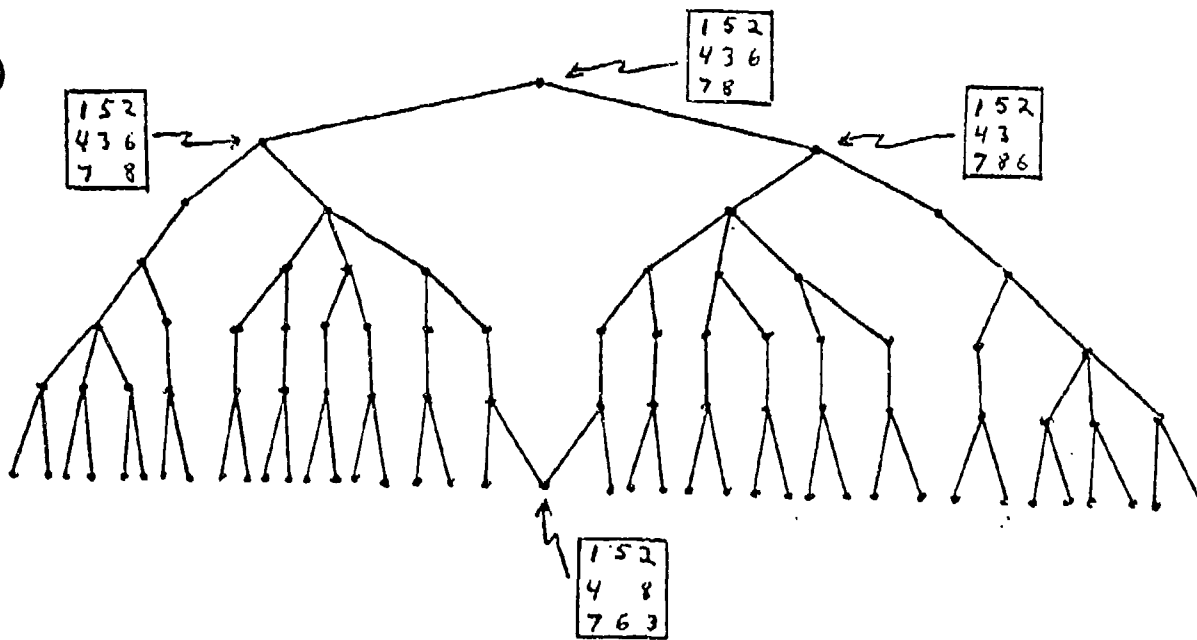


Figure 2.1-1 A portion of the 8-puzzle graph

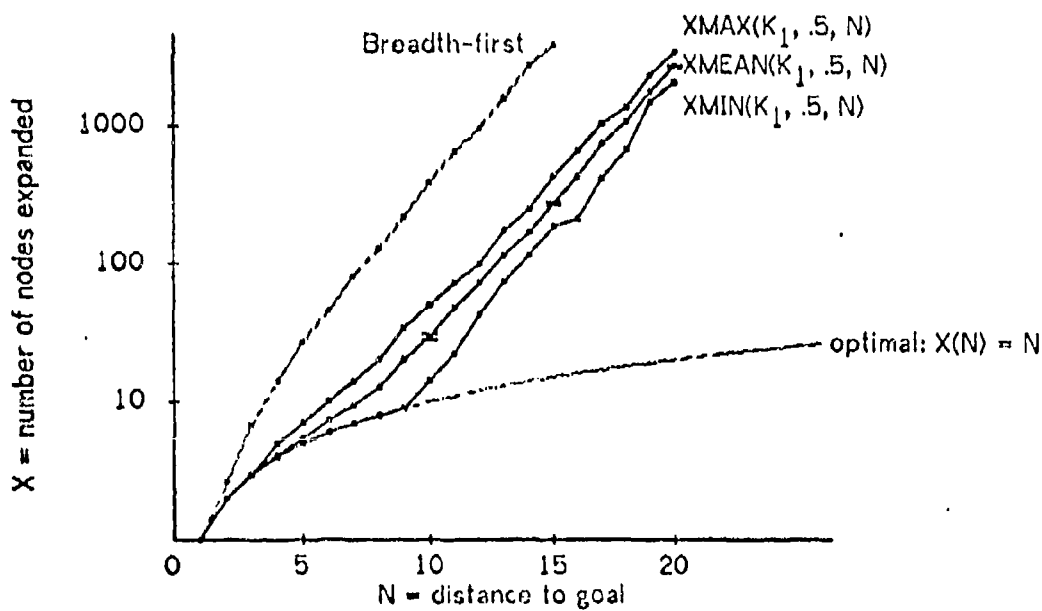


Figure 2.2-1 Mean, maximum, and minimum no. of nodes expanded vs. distance to goal
 A* search of 8-puzzle using heuristic function K_1 , using $W = .5$
 40 samples per data point, 760 samples total (1 sample = 1 problem instance)

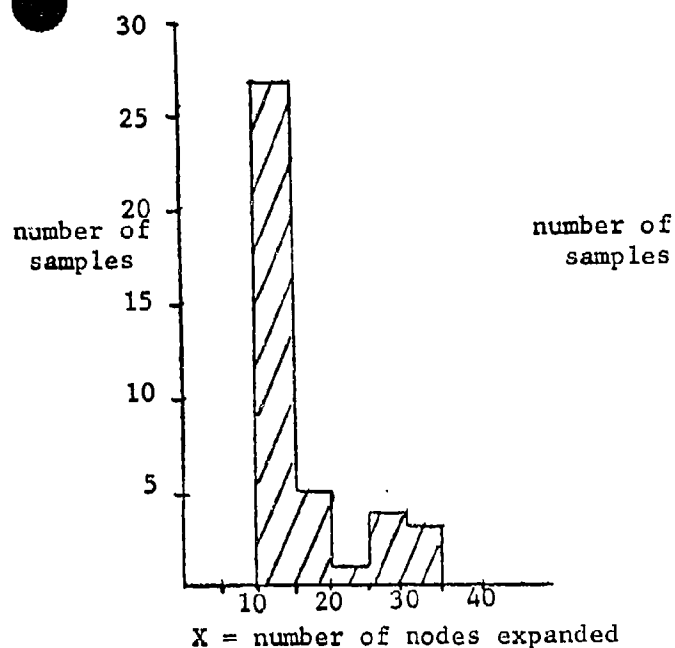
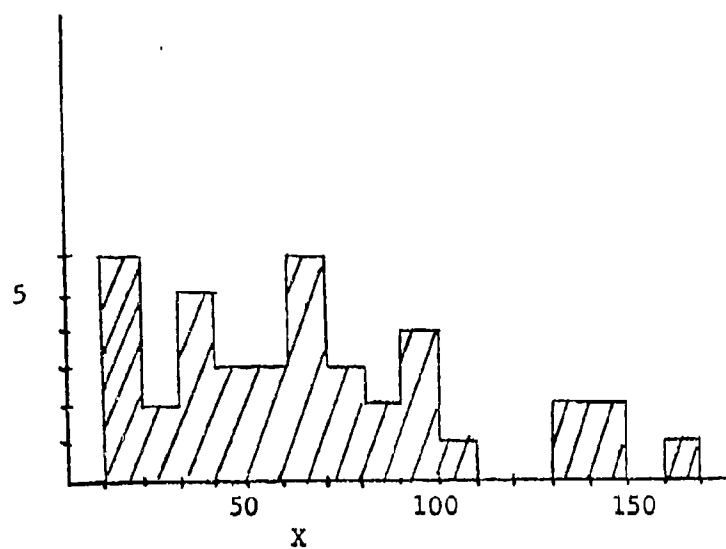
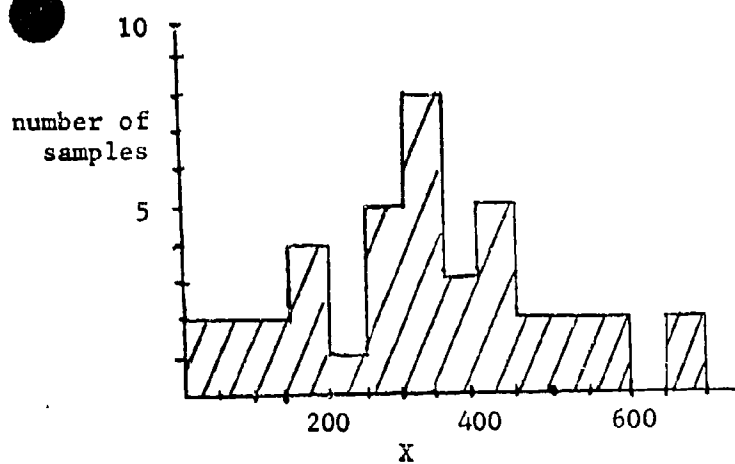
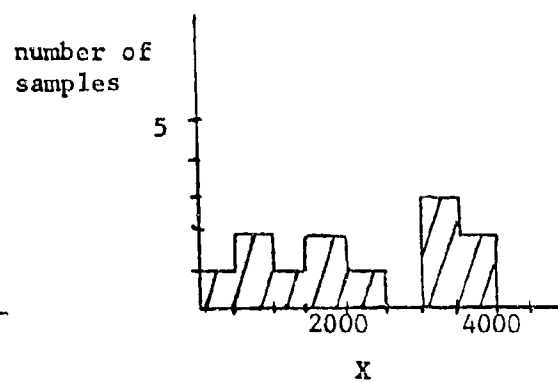
(a) $N = 10$ (b) $N = 15$ (c) $N = 20$ (d) $N = 25$

Figure 2.2-2 Distribution of values of $X(K_2, .5, N)$ for the 8-puzzle for four values of N .

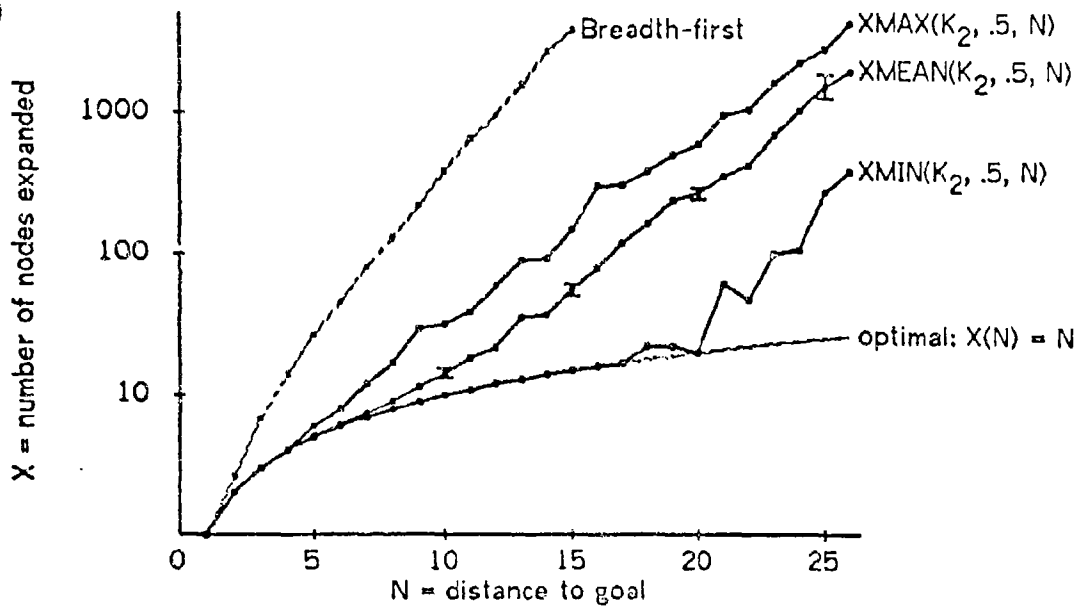


Figure 2.2-3 Analogous to Figure 2.2-1, but using heuristic function K_2
895 problem instances

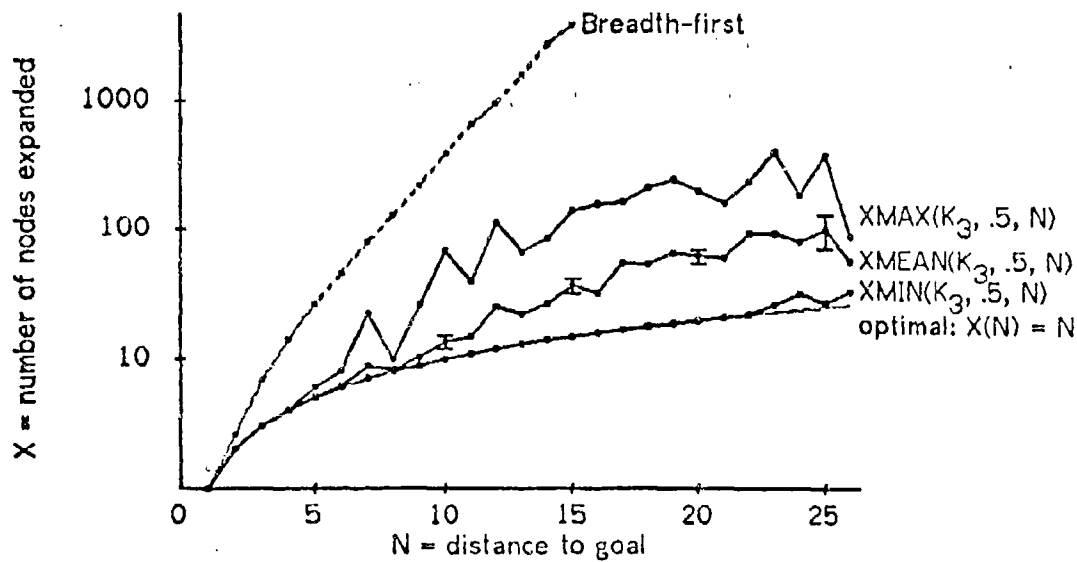


Figure 2.2-4 Analogous to Figure 2.2-1, but using heuristic function K_3
895 problem instances

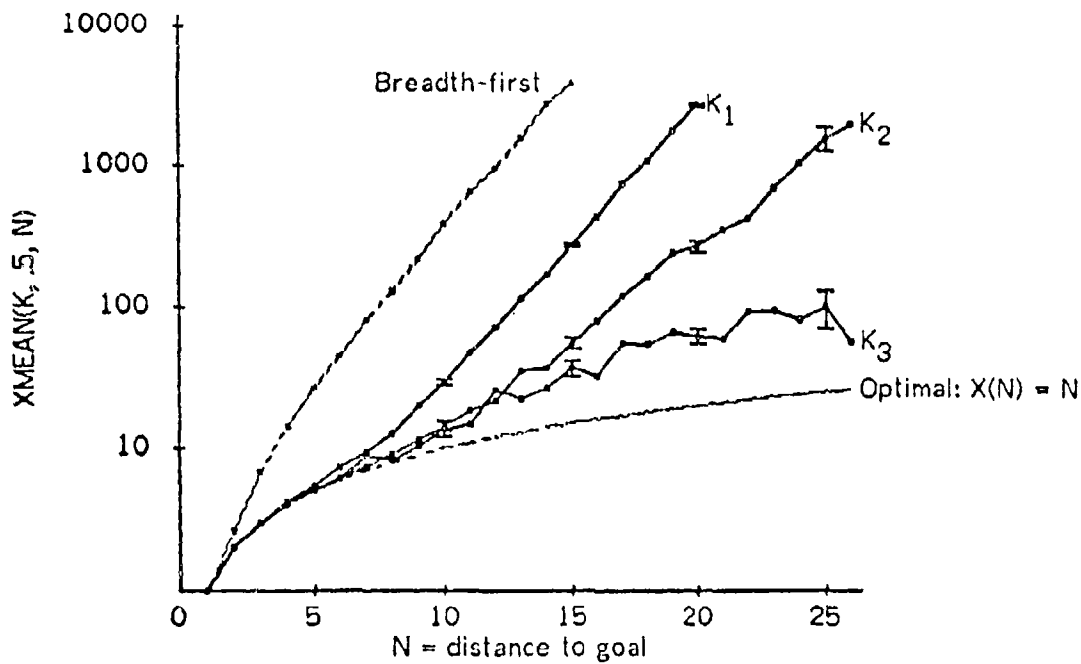


Figure 2.2-5 Mean number of nodes expanded vs. depth of goal,
Data from Figures 2.2-1, 2.2-3, and 2.2-4
760 + 895 + 395 = 2350 algorithm executions

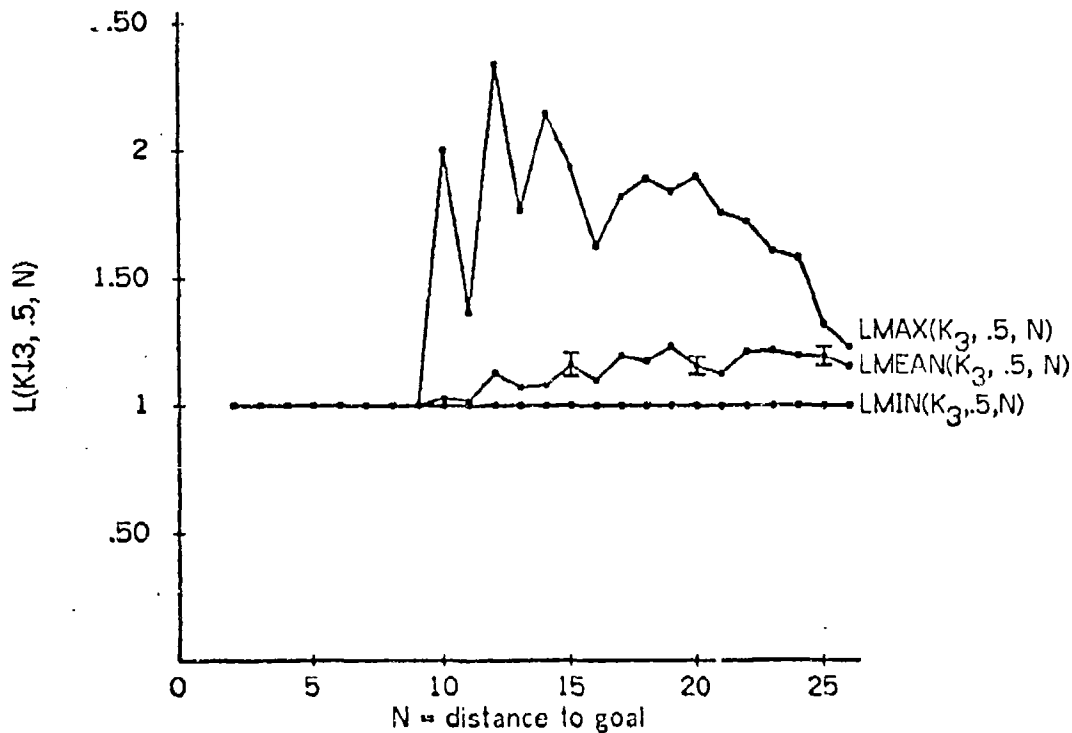


Figure 2.2-6 Length of solution path vs. depth of goal
heuristic function K_3 , using $W = .5$
($L = 1$ if minimal length solution path is found)
895 algorithm executions

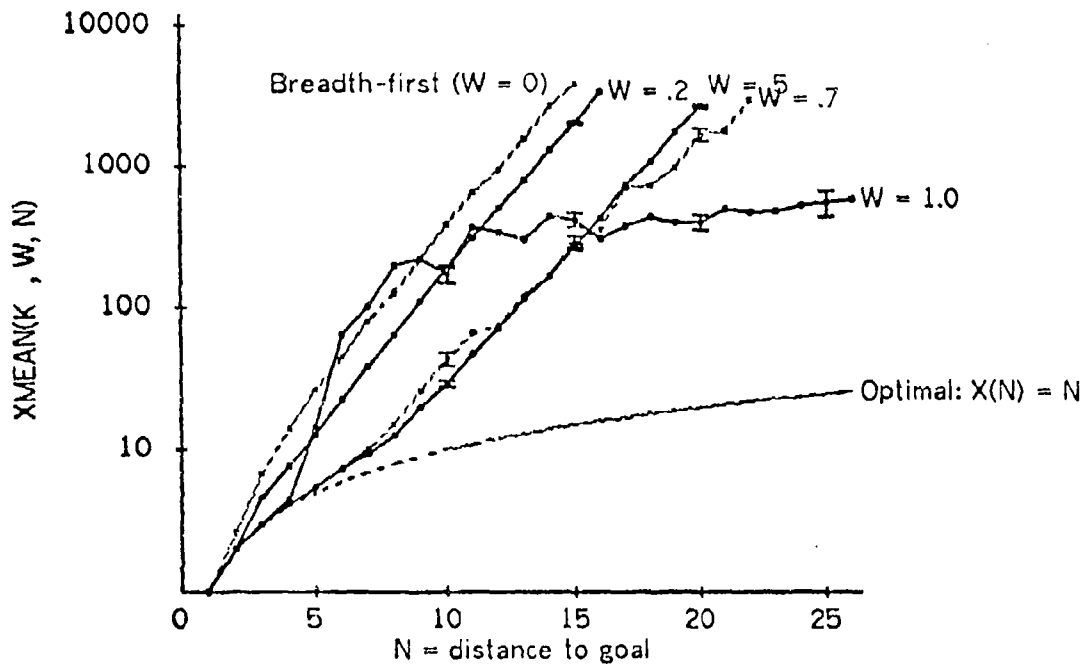


Figure 2.3-1 Mean number of nodes expanded vs. depth of goal
Heuristic function K_1 with different weight values
600 to 895 algorithm executions per value of W

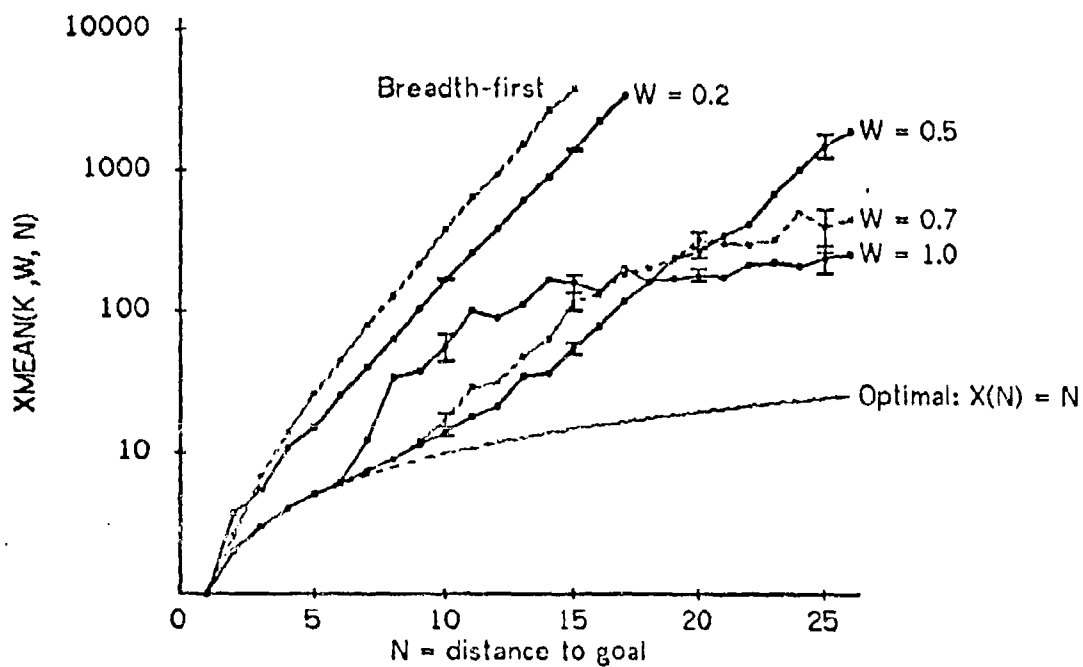


Figure 2.3-2 Analogous to Figure 2.3-1, but for heuristic K_2
640 to 895 algorithm executions per value of W

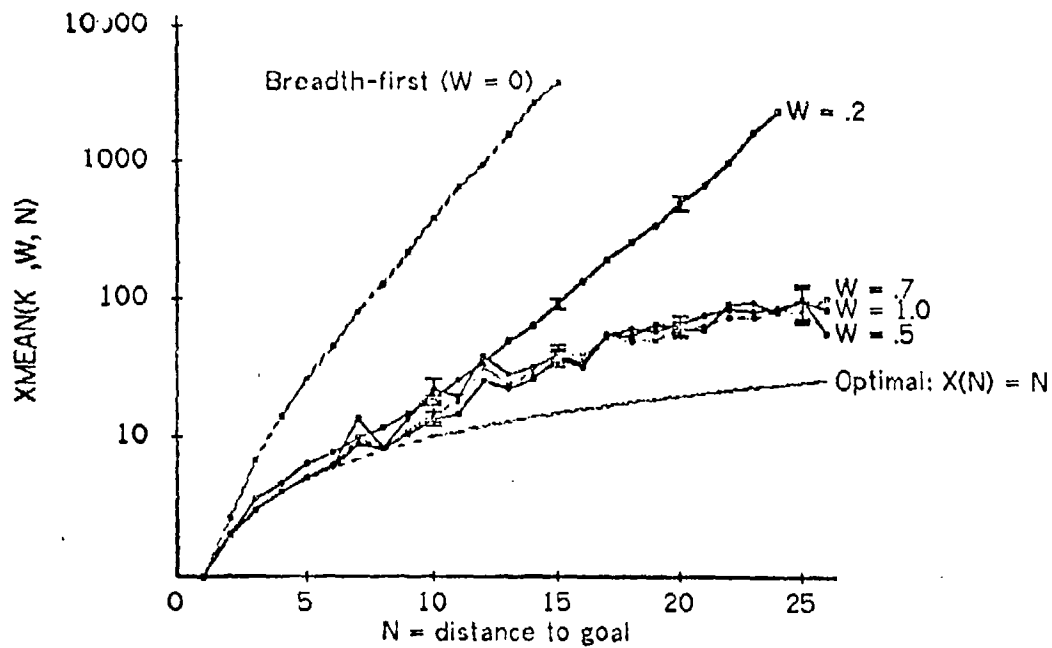


Figure 2.3-3 Analogous to Figure 2.3-1, for heuristic K_3
 These data suggest that increasing W beyond .5 has no apparent effect for K_3
 875 to 895 algorithm executions per value of W

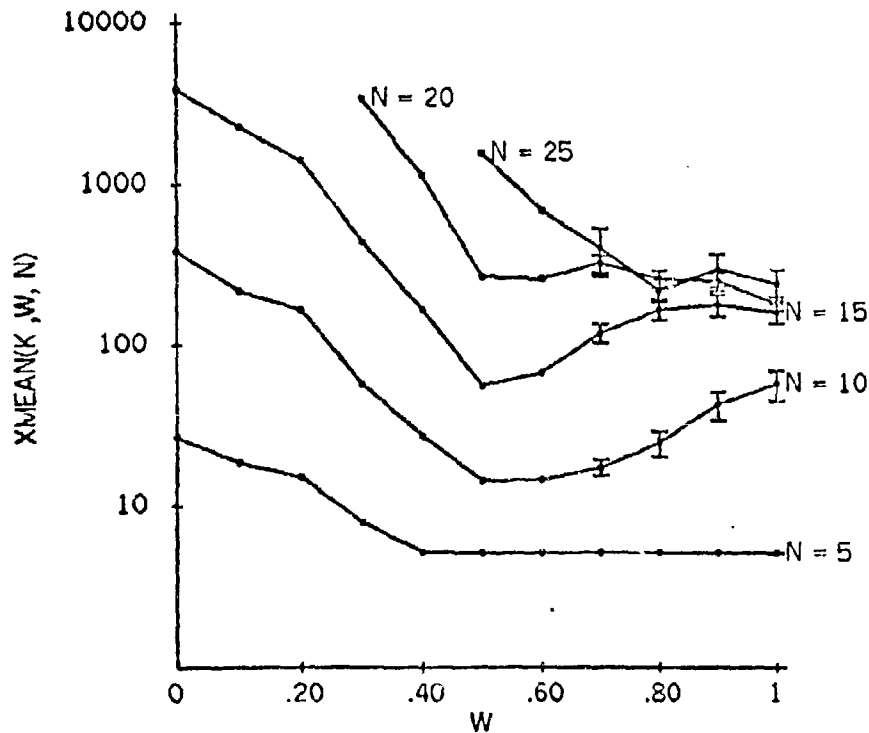


Figure 2.3-4 $XMEAN$ vs. W for K_2 , taken from same data as for Figure 2.3-2

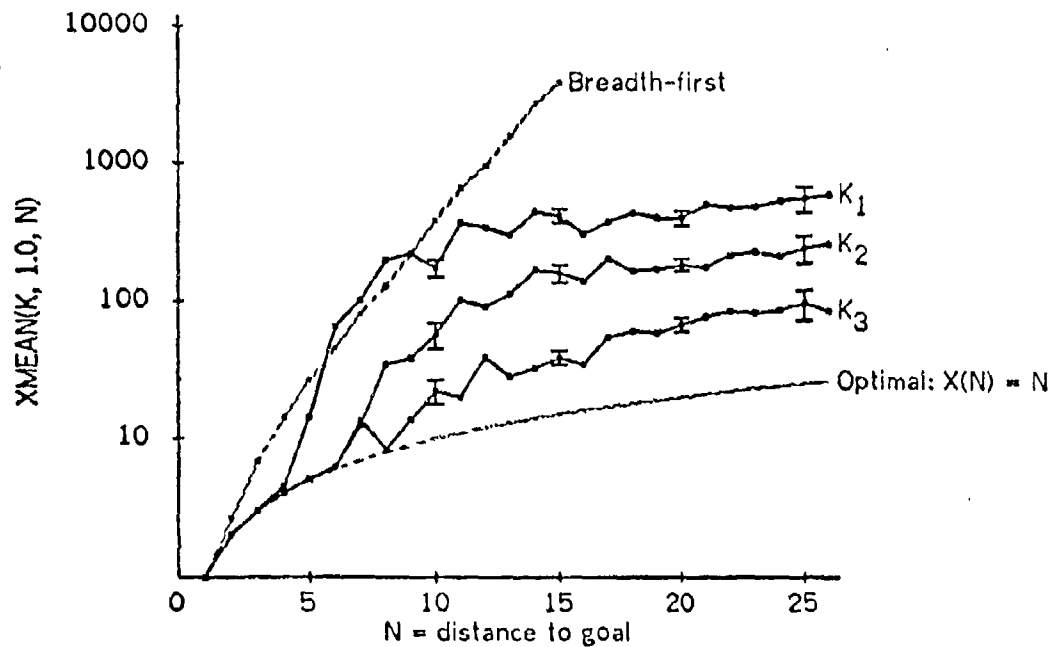


Figure 2.3-5 Mean no. of nodes expanded vs. depth of goal, 3 heuristics using $W = 1.0$
 Combines data for $W = 1.0$ from Figures 2.3-1, 2.3-2, and 2.3-3
 At $W=1.0$, all 3 K functions have "subexponential" cost. Compare with Figure 2.2-5
 $3 \times 895 = 2385$ algorithm executions

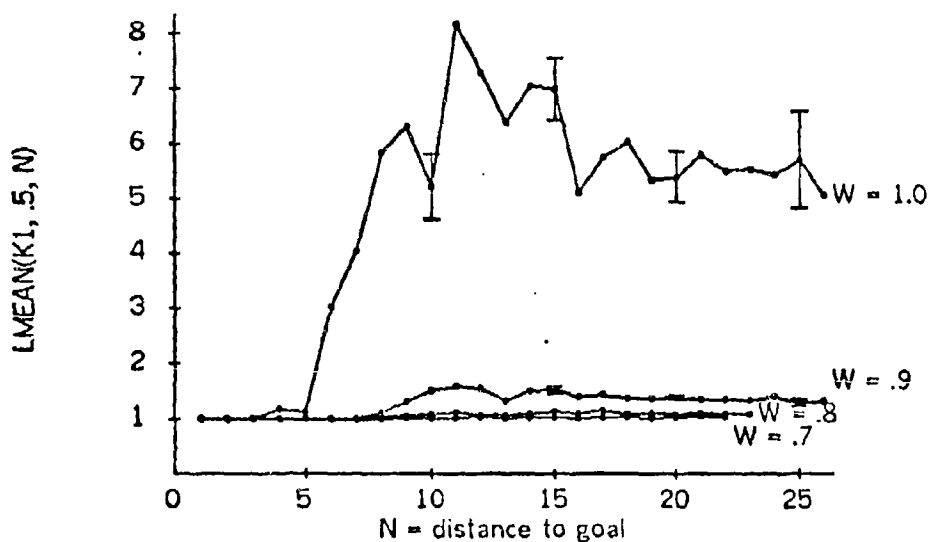


Figure 2.3-6 Mean length of solution path vs. depth of goal, for various W
 heuristic function K_1
 820 to 895 algorithm executions per value of W

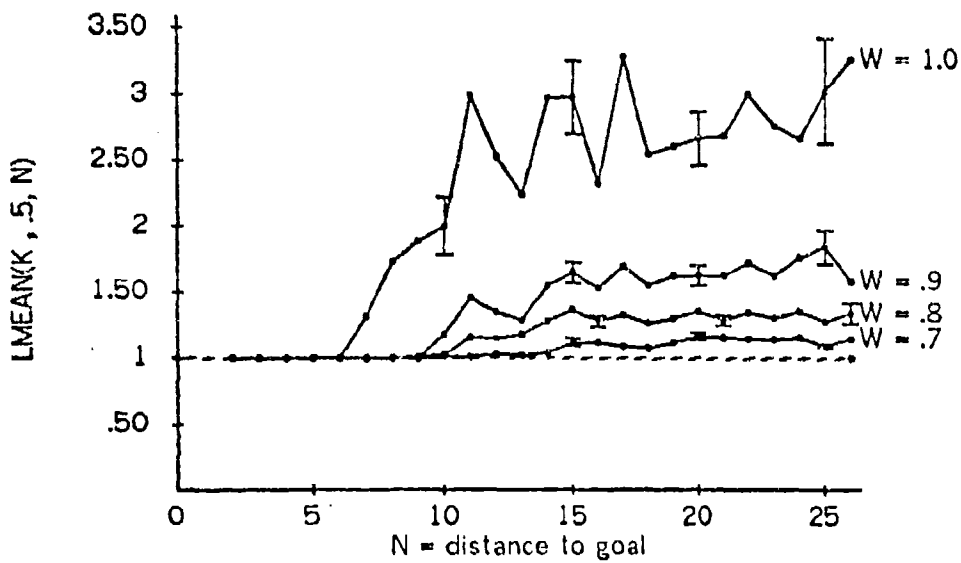


Figure 2.3-7 Mean length of solution path vs. depth of goal, for various W heuristic function K_2
 $4 * 895 = 3180$ algorithm executions

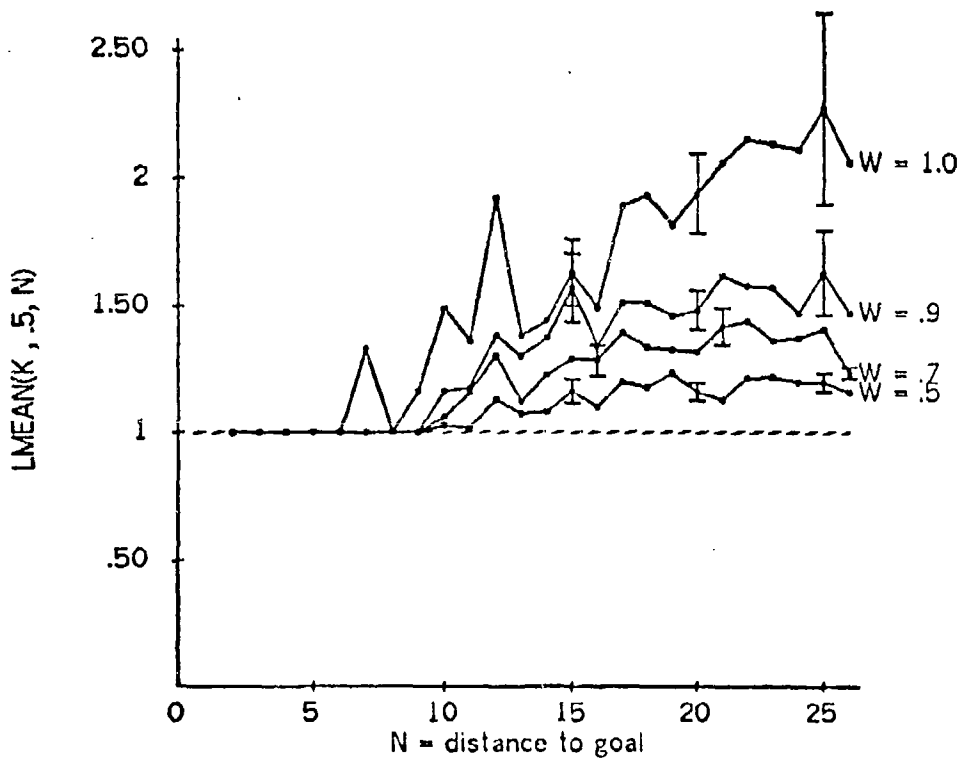


Figure 2.3-8 Mean length of solution path vs. depth of goal, for various W heuristic function K_3
 $4 * 895 = 3180$ algorithm executions

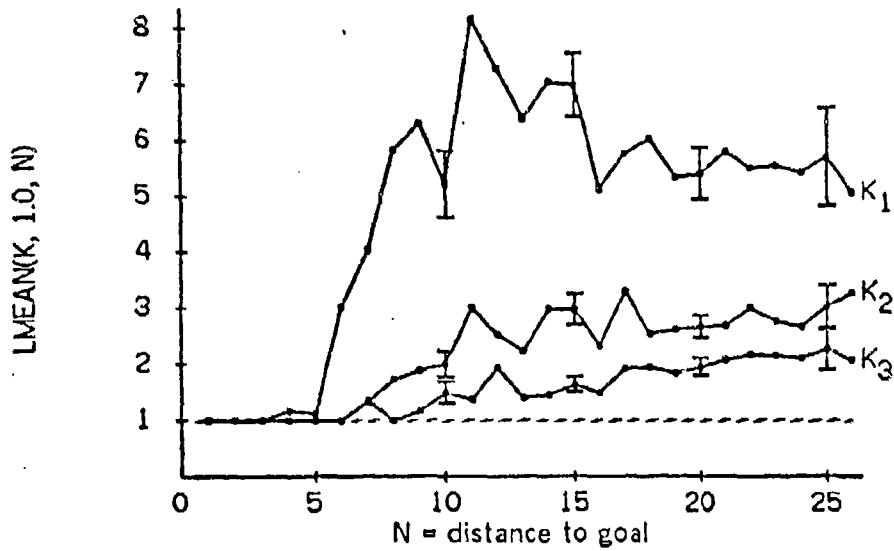


Figure 2.3-9 Mean length of solution path vs. depth of goal, $W = 1.0$
 Comparison of heuristics K_1 , K_2 , and K_3
 (LMEAN = 1 means minimal length solution path was found)
 $3 * 895 = 2385$ algorithm executions

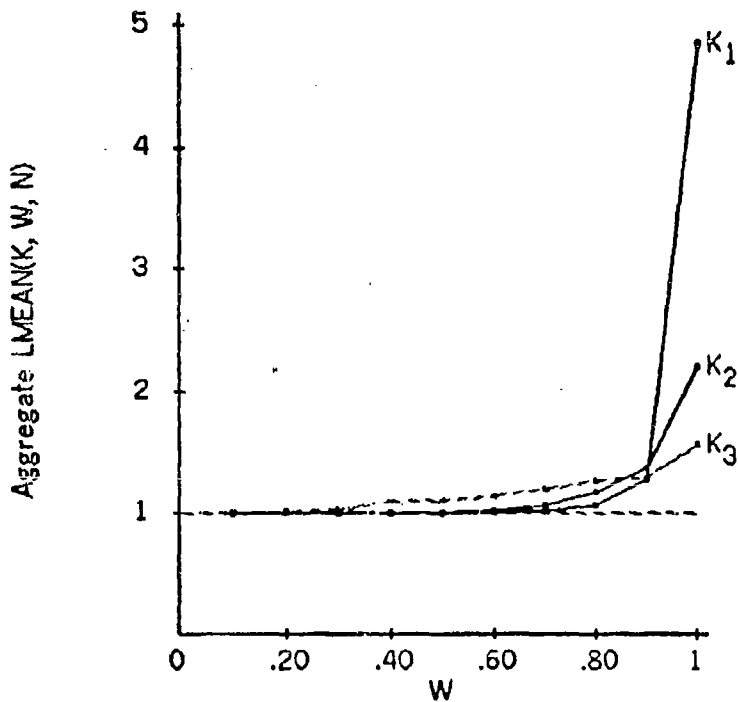


Figure 2.3-10 Aggregate LMEAN(W). See text
 Each data point averages LMEAN over a range of N.
 More than 26,000 algorithm executions

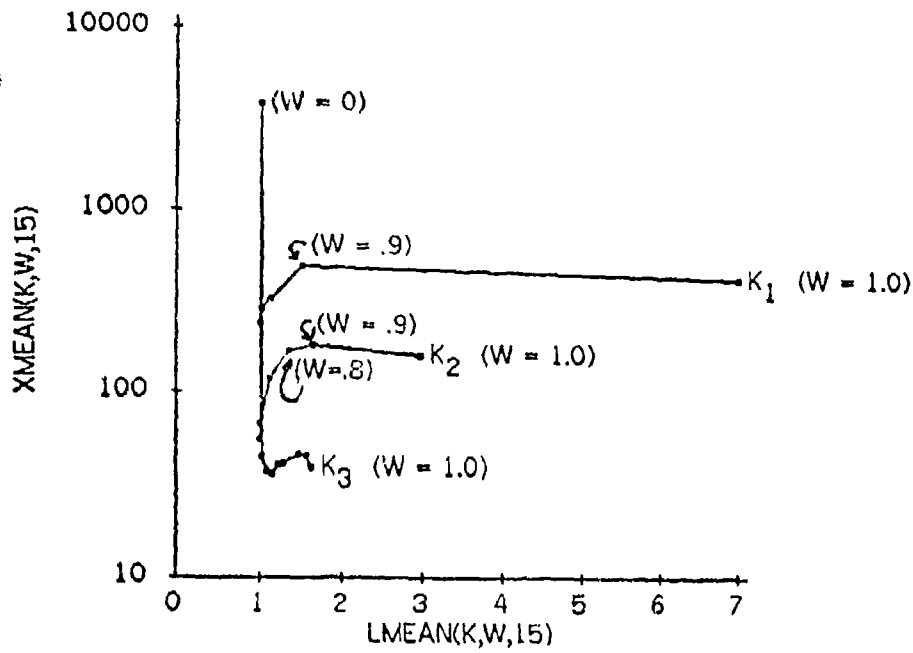


Figure 2.3-11 Cost (XMEAN) vs. quality (LMEAN) over range of W
 $N = 15$

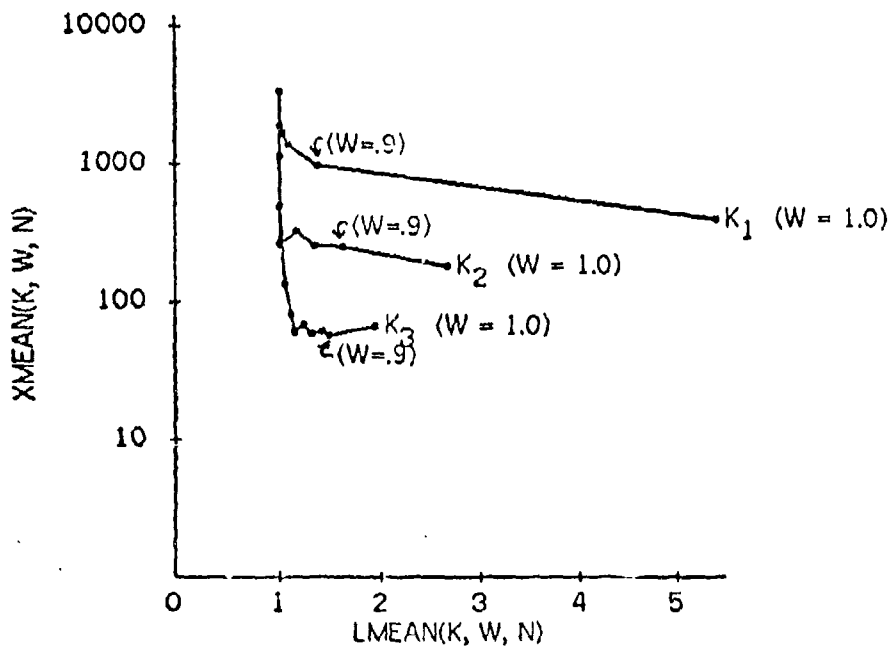


Figure 2.3-12 Cost (XMEAN) vs. Quality (LMEAN) as W varies, $N = 20$
 Analogous to Figure 2.3-11, but for $N = 20$

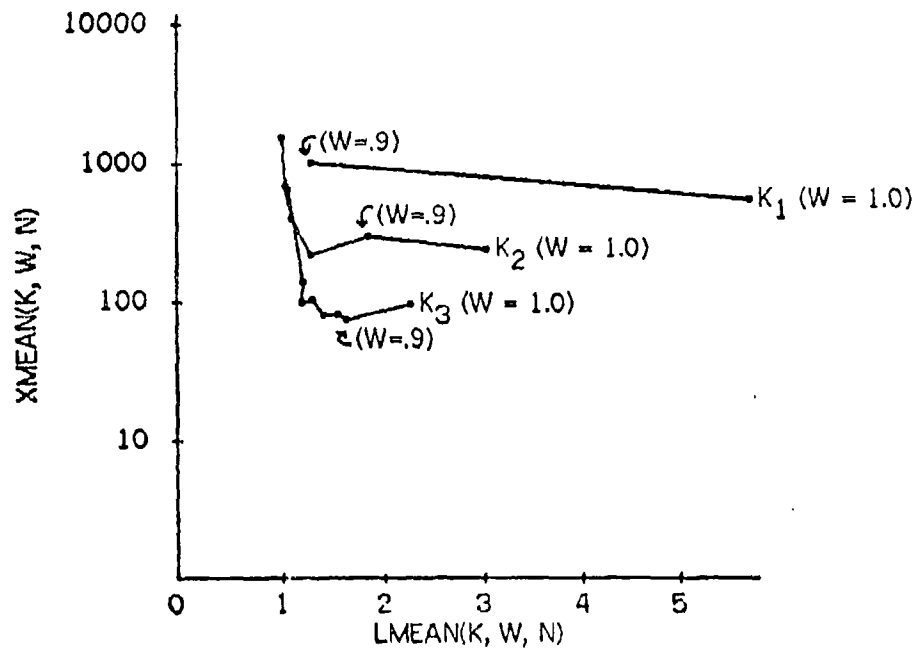


Figure 2.3-13 Cost vs. Quality as W varies, $N = 25$
Analogous to Figure 2.5-11, but for $N = 25$

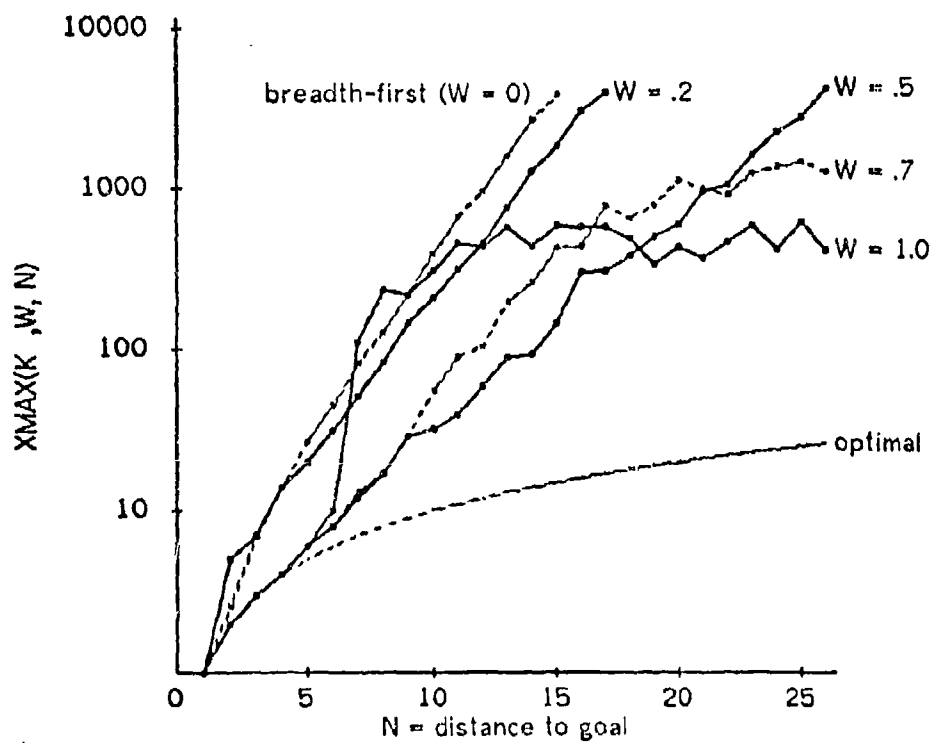


Figure 2.3-14 Maximum number of nodes expanded vs. depth of goal heuristic function K_2 for different values of W
640 to 895 algorithm executions per value of W

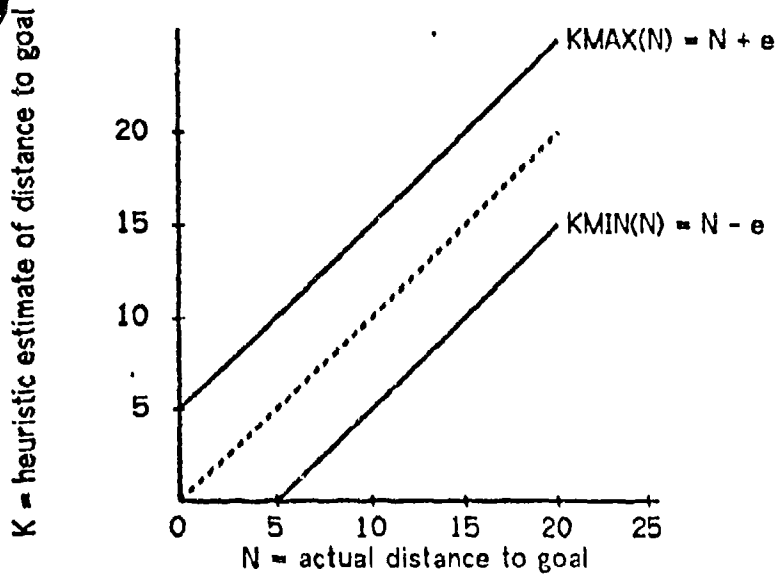


Figure 2.4-1 Heuristic estimate of distance to goal vs. actual distance to goal for the assumptions $K_{MIN}(N) = N - e$, $K_{MAX}(N) = N + e$

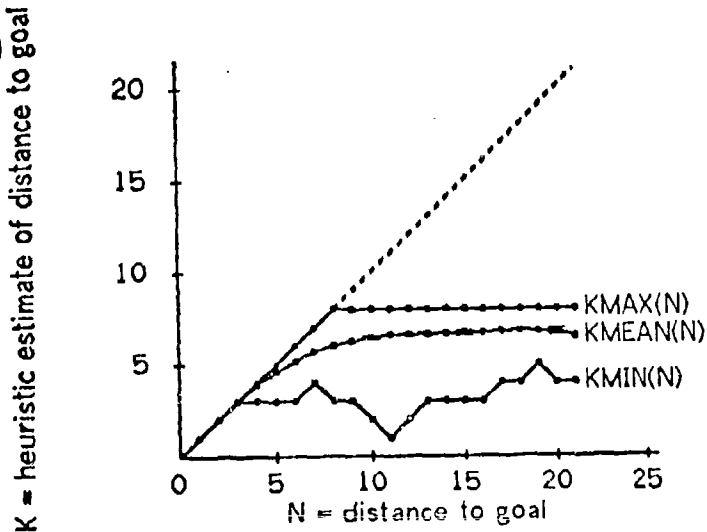


Figure 2.4-2 Estimate of distance to goal vs. actual distance
Heuristic K_1

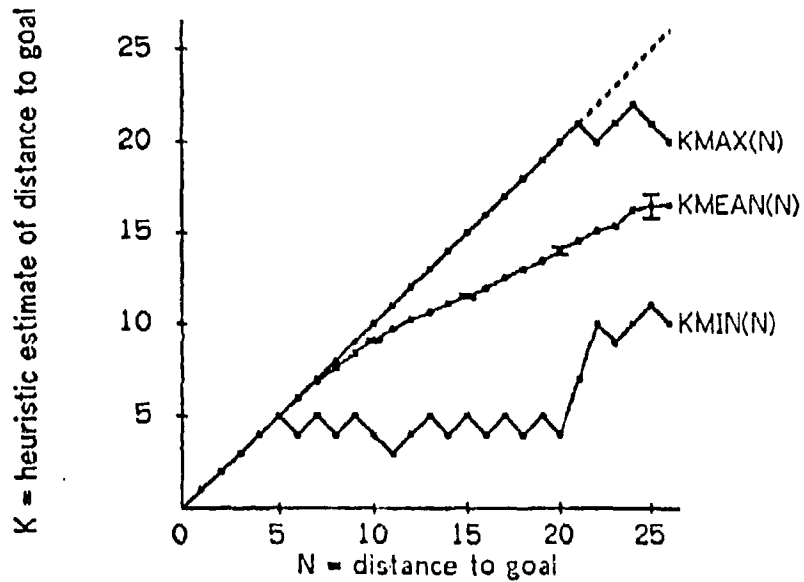


Figure 2.4-3 Estimate of distance to goal vs. actual distance
heuristic K_2
Analogous to Figure 2.4-2

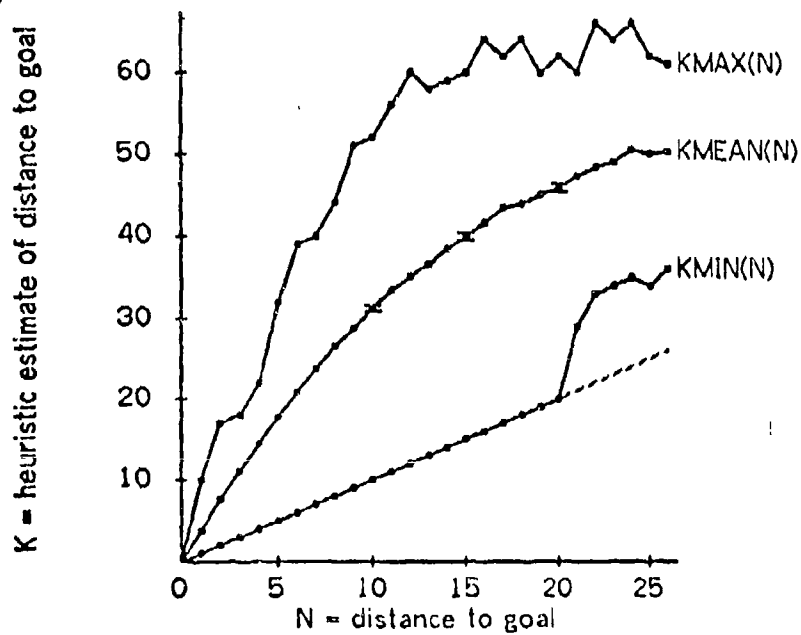


Figure 2.4-4 Estimate of distance to goal vs. actual distance
heuristic K_3
Analogous to Figure 2.4-2

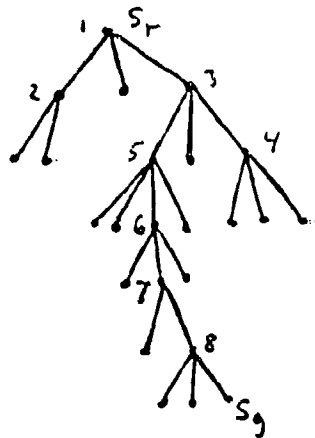


Figure 2.5-1 Hypothetical A^* search illustrating computation of $RUN(G, K, W, s_r, s_g)$. Numbers indicate order of node expansion.

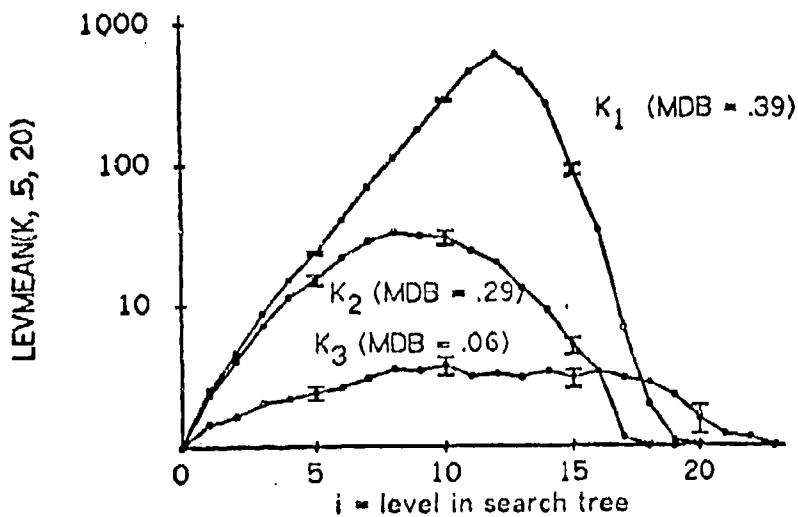


Figure 2.5-2 Mean number of nodes at level i of search tree
Depth of goal = $N = 20$, $W = .5$
 K_1 and K_2 have large "mid-depth bulge" (MDB)

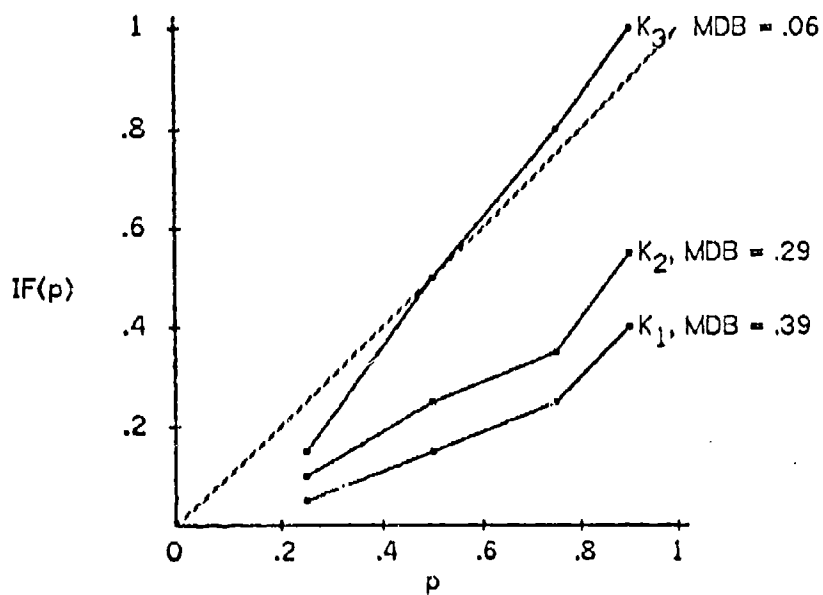


Figure 2.5-3 LEVMEAN interval fraction function for data of Figure 2.5-2
 MDB = mean value of $|LIF(p) - p|$

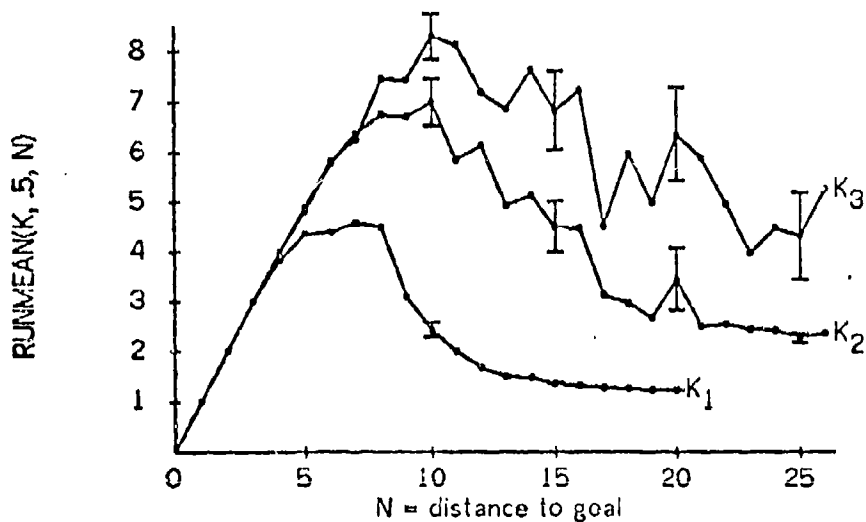


Figure 2.5-4 Mean run length during search
 3 heuristics for 8-puzzle, $W = .5$
 760 to 895 algorithm executions per curve

Chapter 3

Worst Case Cost of A* as a Function of Error in Heuristic Distance Estimates

3.0 Summary of Chapter

In this chapter we determine analytically how the efficiency of the A* best-first search algorithm varies as a function of the heuristic that is used to guide the search. Generalizing the results of Pohl and others from a class of "bandwidth" or "constant absolute error" heuristic functions to arbitrary heuristic functions, we report a worst case cost analysis for A* search of uniform trees in which there is a single goal node at level N . The basic idea of the analysis is that bounds on the distance-to-goal estimates computed by the heuristic map to bounds on the number of nodes expanded, giving cost as a function of N , of the branching factor M , of the estimate-bounding functions representing the heuristic, and of a scalar weighting coefficient W . Previous results were not general enough to predict numerically the number of nodes expanded for heuristic functions used in practice (e.g., heuristics for the 8-puzzle). The present results are general enough to make such predictions. The comparisons of analytic predictions with experimental observations that we make using three 8-puzzle heuristic functions as case studies serve as a basis for assessing quantitatively how realistic the assumptions imposed in our model are.

Section 3.1 defines a search tree and notation identifying the location of any node in the tree. Any heuristic function $K(s)$ on the uniform tree (or on an arbitrary problem graph such as the 8-puzzle) is characterized by two real-valued functions $KMIN(i)$ and $KMAX(i)$ on the non-negative integers that bound the estimates computed by the heuristic function at nodes that are distance i from the goal. Hence the $KMIN$ and $KMAX$ functions given in Chapter 2 for three 8-puzzle heuristic functions have images in the present model (which we call the "DEBET" model, an acronym for "Distance-Estimating, Bounded-Estimate Tree search" model). The definition of the worst case model is completed by assuming that the estimates of distance to goal are determined by the $KMAX$ function for nodes on the solution path, and by the $KMIN$ function for nodes off the solution path, hence favoring the latter at the expense of the former. The cost function $XWORST(M, KMIN, KMAX, W, N)$ gives the number of nodes expanded, and is expressed in terms of an intermediate function $YMAX(KMIN, KMAX, W, r)$ that tells how far from the solution path the search may wander in the worst case (Theorems 3.1-1 and 3.1-2). The resulting formula applies for arbitrary $KMIN$ and $KMAX$ functions, but it is unsatisfying because it is not in closed form and it is difficult to derive any theoretical insight from it.

Section 3.2 derives a simpler formula for the $XWORST$ cost function for a restricted yet realistic class (called IM/DM) of $KMIN$ and $KMAX$ functions (Theorem 3.2-4), and we show how cost varies monotonically with the difference between $KMIN$ and

KMAX (Theorems 3.2-5, 3.2-6, and 3.2-7). The results are illustrated graphically for the special case of "linearly-bounded" heuristics, in which a heuristic function may be represented by a point in part of the Euclidean plane, and cost as a surface in 3-space above the plane. Section 3.3 redefines KMIN(i) and KMAX(i) equivalently in terms of "relative error" $\delta(i)$ and "mean value" $\omega(i)$ functions and establishes the dependence of cost on relative error for the IM/DM class (Theorem 3.3-1). Then for a subclass of "IM-never-overestimating" KMIN and KMAX functions, we derive a very simple formula stating that cost is bounded by a certain exponential function of the relative error in the heuristic's estimates (Theorem 3.3-2). By this formula, we identify heuristics whose cost functions are bounded above by linear, polynomial and exponential functions of N . We also show that cost grows monotonically with relative error, both for the IM-never-overestimating class (Theorem 3.3-3), and for any pair of IM heuristic functions having identical $\omega(i)$ and differing $\delta(i)$ (Theorem 3.3-4).

Section 3.4 derives two results concerning the value of "insurance" terms in an evaluation function, as first posed by Pohl. (Equivalently, these results determine how cost varies if the heuristic function is multiplied by an arbitrary scalar, with the other term held constant.) The first result determines the optimal weighting of the distance-to-goal term and the distance-from-root term in the evaluation function: for the IM-never-overestimating class of heuristic functions, equal weighting is optimal (Theorem 3.4-1). Second, for the class of linearly-bounded heuristics we identify the locus for which an evaluation function consisting of a distance-to-goal term alone is better than one containing a distance-from-root term as well. We determine the difference in cost and plot the results graphically.

Using a numeric approach in Section 3.5, quantitative model predictions based on Theorems 3.1-1 and 3.1-2 are compared to experimental performance measurements of heuristic search of the 8-puzzle, showing good agreement in some cases and agreement within a factor of 10 in most cases. However, the cases of disagreement are themselves revealing: under certain conditions extreme worst case performance does not appear to occur in practice. By quantifying and measuring the extent of agreement, we conclude that the current model is too simple to have practical use, even for the 8-puzzle. Section 3.6 discusses the limitations of the present results. Section 3.7 comments on possible issues involved in defining a mathematical theory that determines how the performance of a "knowledge engine" varies as a function of the "knowledge" it is given.

Summarizing, this is the first analytic worst case cost model of A^* heuristic search which is general enough to claim that "heuristic knowledge" (in a restricted technical sense) is one of the independent variables, and whose predictive applicability in practice is actually tested by direct experiment with familiar (albeit relatively simple) problems.

3.1 A Distance-Estimating, Bounded-Estimate Tree Search Model (DEBET)

3.1.1 Introduction

This chapter analyzes a worst case mathematical model, which we call the DEBET model, of the A^* best-first search algorithm, which solves path-finding problems defined in terms of weighted finite directed graphs. Chapter 2 determines for a case study, the 8-Puzzle, how the performance of A^* varies as a function of the depth of the goal node, the heuristic function used to guide the search, and the value of a weighting coefficient. Here we derive formulas for one performance measure, the number of nodes expanded as a function of the same three parameters, ignoring other performance measures such as length of the solution path found. Since the DEBET model analyzed in this chapter is a simplification of the computational model on which the experimental results of Chapter 2 are based, measurement of the differences between observed values and predicted values serves as a basis for assessing how realistic the present model assumptions are. Previous worst case cost analyses of A^* [Pohl 1970a], [Pohl 1970b], [Nilsson 1971], [Munyer & Pohl 1976], [Munyer 1976], [Vanderbrug 1976], [Pohl 1977] offer few or no numeric tests of this sort.

We set two criteria for success: generality and predictive power. To satisfy the first criteria, we seek a general formula so that the cost of search using any particular heuristic can be determined simply by "plugging in" that heuristic as a parameter to the formula, and evaluating it. The key issues here are a) how large and how representative a set of heuristics can be spanned by a single formula, and b) how simple is that formula. To satisfy the predictive power criteria, the model assumptions necessary to permit tractable analysis must be realistic enough that the model's quantitative predictions do agree, to some measurable extent, with experimental observations of the more complex phenomena that are modelled.

The previous A^* worst case cost analyses cited above fail to meet the generality criteria: results are restricted to a very simple class of heuristics, a class that does not include (as shown in Chapter 2) heuristics typically used in practice, even for relatively simple problems like the 8-puzzle. By generalizing these previous results to the case of arbitrary heuristics, the current results permit the validity of the model

assumptions to be tested experimentally for realistic (albeit relatively simple) problems.

Besides the objective of sufficient generality to permit numerical predictions for realistic cases, we also seek to derive formulas that are intuitively meaningful. As it turns out, the formulas we shall derive for the most general case require certain computations to evaluate. When specialized under certain simplifying assumptions, however, these theorems take a simpler and more meaningful form. One such restricted class of heuristic functions, what we call "linearly-bounded" functions, is discussed at various points throughout this chapter. The bounds on the estimates of distance to the goal computed by a linearly-bounded heuristic function increase linearly with distance from the goal. Each linearly-bounded function can be specified by two scalar values a and b . The results permit a geometric interpretation of cost as a "surface" above the a - b plane.

A first step toward the goal of modeling best-first search of arbitrary problem graphs, for example the 8-puzzle, is to model best-first search of uniform trees. In the remainder of Section 3.1, we describe a way to characterize heuristics mathematically, we determine for an arbitrary heuristic function exactly which nodes are expanded during search, and then give a formula that counts them. The current worst case cost model follows the work of [Hart et.al. 1968], [Pohl 1970a], [Pohl 1970b], [Nilsson 1971]. These earlier results were extended more recently by [Munyer & Pohl 1976], [Munyer 1976], [Vanderbrug 1976], and [Pohl 1977]. Other aspects of the A^* algorithm are considered in [Chang & Slagle 1971], [Harris 1974], [Ross 1973], [Ibaraki 1976], [Martelli 1977], and [Gelperin 1977].

3.1.2 First Definitions

A formalism for heuristic search requires definitions for five things: state graph, search schema, solution condition, heuristic, and cost measure.

Definition 3.1-1.

Let $T(M, N)$ denote the uniform tree of branching factor M and unbounded depth, with one distinguished node at level N (called the "goal" node). The root node is at level 0. M is a positive integer. (See Figure 3.1-1.)

- s denotes a node in $T(M, N)$
- $g(s)$ denotes the level of node s in $T(M, N)$
- $p(s)$ denotes the level of the deepest common ancestor of node s and the goal node. Call this the "depth of divergence" of node s .
- u_i denotes the node at level i on the (unique minimal length) solution path from the root to the goal. Thus u_0 is the root node and u_N is the goal node. We refer to the node $u_{p(s)}$ as the "node of divergence" of a node s .
- $r(s) = N - p(s)$, i.e., the distance from the node of divergence of s to the goal
- $y(s) = g(s) - p(s)$, i.e., the distance from the node of divergence of s to s
- $v_s(i)$ denotes the node at level i on the path from the root to s
- SP A node s is SP iff it is a node u_i on the solution path
- NSP A node s is NSP iff it is not SP

SP and NSP are considered alternatively as sets or as predicates, as convenient. The terms $g(s)$ and $p(s)$ are useful in formulating the model, whereas the use of $r(s)$ and $y(s)$ simplifies the analysis.

Figure 3.1-2 illustrates the approach taken in the remainder of this section. Each heuristic function causes certain nodes in $T(M, N)$ to be expanded. In general, different heuristic functions cause different sets of nodes to be expanded, and our objective is to derive a formula for the number of nodes expanded as a function of the heuristic function. The formulation of this worst case model is such that all nodes in certain depth-limited subtrees are expanded, as suggested in a vague way in Figure 3.1-2. To count the total number of nodes in $T(M, N)$ that are expanded when using an arbitrary heuristic function, we derive a formula for the depths of these subtrees (i.e., the function Y_{MAX}). Then it is a simple matter (by Theorem 3.1-2) to express the total number of nodes expanded in terms of the values of Y_{MAX} .

Distance from node s to the goal node is denoted $h(s)$, the distance from s to its node of divergence (i.e., distance back to the solution path) plus the distance from there to the goal (distance along the solution path). Formally,

$$\begin{aligned} h(s) &= N - p(s) + g(s) - p(s) \\ &= y(s) + r(s) \end{aligned} \quad (3.1-1)$$

Note that if s is SP, (3.1-1) reduces to the formula $h(s) = N - p(s) = r(s)$. Note also that there are in general many nodes having a given value of $h(s)$. For example, node u_{N-4} is distance 4 from u_N ; the $M - 1$ nodes having $r(s) = 3$ and $y(s) = 1$ are also distance 4 from u_N , and so are the $M \cdot (M - 1)$ nodes having $r(s) = 2$ and $y(s) = 2$, and so on, as illustrated by Figure 3.1-3. $R_{M,N}(i)$ denotes the set of those nodes in $T(M,N)$ that are at distance i from the goal, i.e., the nodes for which $h(s) = i$.

The root node of the tree is taken to be the initial node of the search. The search terminates when the distinguished node at level N is selected for expansion. Note that when searching graphs, two possible solution criteria can be specified: (a) find any path between root node and goal node, and (b) find a path of minimal length. In a tree, the solution path is unique, and so the present model forgoes any possibility of deriving results concerning the conditions under which solution paths are or are not of minimal length. Experimental measurements of the lengths of the solution paths for search of the 8-puzzle under a variety of conditions are given in [Doran & Michie 1966] and in Chapter 2.

Note that the current analysis ignores completely the question of how difficult it is to compute the value of a particular heuristic function in practice, and ignores the form in which the function is represented in practice. (Such questions are of course important, but a rigorous treatment is problematic.) For purposes of the present analysis, a heuristic function simply computes some definite values, i.e., a heuristic is simply a mathematical function. Hence the model results take the form: if the values computed by the heuristic happen to be such and such, then the number of nodes expanded is thus and so. Our approach thus is to define a class of mathematical functions such that a heuristic that occurs in practice corresponds to some function in the class.

An evaluation function F assigns a value to each node in $T(M,N)$. Given any function $F: S(M, N) \rightarrow \mathbb{R}^+$ (where $\mathbb{R}^+$ throughout denotes the non-negative reals, and

$S(M,N)$ denotes the set of all nodes in $T(M,N)$, excluding the descendants of the goal node u_N , A^* is defined as follows.¹

Algorithm A^* (for trees):

1. Mark u_0 as "OPEN" and compute $F(u_0)$.
2. Choose an OPEN node s whose F value is minimal, resolving ties arbitrarily.
3. If s is the goal node u_N , then terminate.
4. Mark s as "CLOSED", and compute $F(v_i)$ for each son node v_i of s . Mark each such node as OPEN. Go to step 2.

An execution of step 4 is said to "expand" node s .

In practice, the value of N is not known a priori; step 3 is implemented typically as a predicate that determines whether the current node s satisfies the goal condition. For any particular search, however, N has some definite value. The current analysis likewise makes no a priori assumptions about the value of N , but rather gives its answers in the form: if the value of N happens to be such and such, then the number of nodes expanded is thus and so.

3.1.3 A General Case Theorem: Which Nodes are Expanded?

Best first search can be understood intuitively in terms of a "filling the valleys" metaphor. Imagine the search tree as a geographical terrain which one enters at a designated entry point (the root node). One traverses the terrain by following any of the foot paths laid out on it, which form a tree structure. The object is to find a path to a particular junction along one of the paths, at which a treasure is located. The treasure cannot be seen from a distance, but is evident when one arrives at the

¹ Footnotes 6 and 9 in Chapter 2 explain our rationale for using the symbol F to denote what [Hart et al. 1968] and others call $\hat{f}$. Similarly, what we subsequently call K , these others call $\hat{h}$. In searching trees, the length of the path found during a search from the root node to a given node s (the quantity these others call $\hat{g}(s)$) always equals $g(s)$, the distance in the tree from the root node to node s .

junction marking its location. The terrain is not flat; instead, it is rather mountainous; a junction may be higher or lower in elevation than its neighbors.

A best-first search always seeks minima. If the path to the goal must cross a hill but there is a side path along the way leading to a valley, the valley will be completely explored before proceeding again upward. A* best-first search provides the ability to hop to any junction previously visited, if one of its yet unvisited neighbors is now the lowest in elevation of all such candidates (including those of the current junction). The evaluation function $F(s)$ defines the elevation of each junction s in the tree-structured terrain. The root node and goal node and tree structure of paths are fixed, but different heuristics define different terrains thereupon.

As in the "filling the valleys" metaphor, in A* search the absolute values computed by F are irrelevant; the relative ordering of the values suffices to determine whether or not a node is expanded, and the order in which nodes are expanded. The following theorem identifies in the most general case which nodes are expanded, as illustrated in Figure 3.1-4.

Theorem 3.1-1. Let $F: S(M, N) \rightarrow \mathbb{R}^+$ be arbitrary, subject to the condition that A* search of $T(M, N)$ terminates when F is used as evaluation function. Assume conservatively that ties in step 2 of A* are always resolved in favor of a NSP node. Then any NSP node s is expanded at some time before termination iff

$$\max_{i=p(s)+1}^{g(s)} F(v_s(i)) \leq \max_{j=p(s)+1}^N F(u_j) \quad (3.1-2)$$

Proof sketch. ad sufficiency:

Let c be the smallest value of j such that $p(s) < j \leq N$ and $F(u_j) = \max_{i=p(s)+1}^N F(u_i)$. Since A* terminates, it follows that every SP node is selected (at step 2) for expansion, and hence is OPEN prior to expansion, so in particular u_c is OPEN sometime before A* terminates. Nodes $v_s(p(s)+1)$ and $u_{p(s)+1}$ have the same father node (namely $u_{p(s)}$), hence are OPENed at the same step. Hence either $v_s(p(s)+1)$ is OPEN prior to the time u_c is OPENed, or $c = p(s) + 1$ and hence u_c and $v_s(p(s)+1)$ are OPENed at the same step. By assumption $F(v_s(p(s)+1)) \leq F(u_c)$, hence node $v_s(p(s)+1)$ is expanded before node u_c , if at all. Since u_c is expanded it follows that $v_s(p(s)+1)$ is expanded. When $v_s(p(s)+1)$ is expanded, $v_s(p(s)+2)$ is OPENed. Hence the latter is OPENed before u_c is expanded. By assumption $F(v_s(p(s)+2)) \leq F(u_c)$, consequently $v_s(p(s)+2)$ is expanded. Continuing the argument for nodes $v_s(p(s)+i)$ for $i > 2$, it follows by induction on i that

node s is expanded.

ad necessity: The converse follows by similar argument and is omitted here. $\square$

Theorem 3.1-1 is fundamental to what follows. Theorem 3.1-1 identifies which nodes in $T(M,N)$ are expanded by A^* for any given values of M , N , and F .² In the sequel, Theorem 3.1-2 gives a formula for counting the number of such nodes, and this formula is expressed in terms of an intermediate function $YMAX$. Lemmas 3.2-1, 3.2-2, and 3.2-3 permit Theorem 3.2-4, which for certain restricted F functions gives a simpler formula for $YMAX$. The remaining theorems build upon them.

3.1.4 The $\langle KMIN, KMAX \rangle$ Model of Heuristic Functions: Definitions

In Theorem 3.1-1, the function F takes a general form. In what follows, F always assumes the particular form

$$F(s) = (1-W) g(s) + W K(s) \quad (3.1-3)$$

where $K: S(M,N) \rightarrow \mathbb{R}$, and W is a real number such that $0 < W \leq 1$. (The degenerate case $W = 0$ corresponds to a breadth-first search.) If $K(s)$ is interpreted as an estimate of $h(s)$, then $F(s)$ as in (3.1-3) is a linear combination of the distance from the root node U_0 to s and the heuristic estimate of distance from s to the goal node u_N . The results given in Chapter 2 and in [Nilsson 1971, pp. 54-77] motivate the study of this form. A^* terminates for any $F(s)$ satisfying (3.1-3) such that $W < 1$ because the presence of the $g(s)$ term insures that any infinite path in $T(M, N)$ must contain a node s for which $F(s)$ exceeds the maximum of the $F(u_i)$ values, for $i = 0, 1, \dots, N$.

² The proof of Theorem 3.1-1 also provides insight about the set of $F(s)$ functions for which A^* fails to terminate. Recalling that $T(M,N)$ is a uniform tree of unbounded depth, A^* search using evaluation function F will fail to terminate if and only if one of the following conditions hold: (1) $F(s) \leq F(u_1)$ for all of the NSP nodes s having $p(s) = 0$; (2) $F(s) \leq F(u_2)$ for all of the NSP nodes s having $p(s) = 1$; ...; (N) $F(s) \leq F(u_{N-1})$ for all of the NSP nodes s having $p(s) = N-1$. We implicitly exclude such degenerate F functions from consideration.

The basic idea of the analysis that follows is that bounds on the estimates computed by a heuristic function map to bounds on the cost of search using that function. The classes of bounding functions $KMIN(i)$ and $KMAX(i)$ are defined, respectively, as follows. A heuristic function $K(s)$ assigns a value to each node s in the tree $T(M,N)$. In general there are many nodes in the tree at distance i from the goal node. Hence the values assigned by K to the nodes at distance i from the goal will in general span a range of values, from what we call $KMIN(i)$ to $KMAX(i)$. This is expressed formally as follows.

Definition 3.1-2.

- a) Given values for M and N , let KS denote the set of all functions of the form $K: S_{M, N} \rightarrow \mathbb{R}^+$.
- b) For any $K \in KS$, let $KMIN(M, N, K, i)$ be the smallest value of $K(s)$ for $s \in R_{M, N}(i)$, and similarly let $KMAX(M, N, K, i)$ be the largest value of $K(s)$ for $s \in R_{M, N}(i)$. (When M, N , and K are known implicitly, we write simply $KMIN(i)$ and $KMAX(i)$.)
- c) Let KB denote the set of all functions from $\mathbb{N}$ to $\mathbb{R}^+$, where $\mathbb{N}$ denotes the non-negative integers, and let $\langle KMIN, KMAX \rangle$ denote an element of $KB \times KB$ such that $\forall i \ KMIN(i) \leq KMAX(i)$. Let KB^* denote the set of all such $\langle KMIN, KMAX \rangle$.

Figure 3.1-5 shows some of the values of an arbitrary K function and of its corresponding $KMIN(i)$ and $KMAX(i)$ functions.³ Since $KMIN$ and $KMAX$ are defined in terms of the distance function $h(s)$, a slightly modified version of Definition 3.1-2 applies to graphs such as the 8-puzzle as well as to uniform trees. Hence any heuristic function $K(s)$ for the 8-puzzle (or for any graph problem of this sort) has a corresponding $\langle KMIN, KMAX \rangle$ within the DEBET model. Figures 2.4-2, 2.4-3, and 2.4-4 in Chapter 2 show experimental measurements of the $KMIN(i)$ and $KMAX(i)$ values corresponding to the 8-puzzle heuristics called K_1 , K_2 , and K_3 in Chapter 2.

³ Note that the tree $T(2,4)$ is unbounded, whereas Figure 3.1-5a shows only the portion at depth 4 or less. In particular, there are some nodes at depth greater than 4 in $T(2,4)$ that are at distance 3 from the goal node, at distance 4, and so on. The $K(s)$ values assigned to these nodes also contribute to determining $KMIN(3)$ and $KMAX(3)$, $KMIN(4)$ and $KMAX(4)$, and so on. For simplicity, the $KMIN(i)$ and $KMAX(i)$ values given in Figure 3.1-5b are based on only the nodes shown in Figure 3.1-5a. The same applies to Figures 3.1-8 and 3.1-9.

The previous A^* worst case cost analyses cited at the beginning of this section assume $KMIN$ and $KMAX$ to have the form $KMIN(i) = i - a$ and $KMAX(i) = i + b$, where a and b are real-valued constants, i.e., the form shown in Figure 2.4-1. This simplifies the analysis, but it excludes heuristics whose $KMIN$ and $KMAX$ are more arbitrary, e.g., those in Figure 2.4-2, 2.4-3, and 2.4-4.

Besides the general theorems 3.1-1 and 3.1-2 (upcoming), we shall also derive simpler and more intuitively meaningful formulas for certain restricted classes of $\langle KMIN, KMAX \rangle$ functions. A brief description now of one such class may help to motivate the subsequent definitions and theorems. The $\langle KMIN, KMAX \rangle$ function shown in Figure 3.1-6 is an instance of what we call a "linearly-bounded" heuristic function. In this case $KMIN(i)$ and $KMAX(i)$ grow linearly with i , the distance to the goal. Hence each such heuristic function can be identified by two scalar values a and b . We denote a linearly-bounded function thus: $\langle a, b \rangle$.

Figure 3.1-7 represents the set of all such linearly bounded heuristic functions as the portion of the Euclidean plane for which $B \geq a$. The point (a, b) in the plane corresponds to the heuristic function $\langle a, b \rangle$, as depicted in the four "blowups" in Figure 3.1-7. (Note that Figure 3.1-7 is drawn to scale.) Associated with each such $\langle a, b \rangle$ function is a corresponding cost function, $XWORST$ (defined subsequently), telling how many nodes that particular function expands as a function of N , the depth of the goal. Hence $XWORST$ defines a sort of cost "surface" above the a - b plane. However, the "height" of each point on this surface is not given by a scalar value, but rather by a function (of N). It turns out for the class of linearly-bounded functions that these cost functions can be mapped to scalar values, so that the cost associated with linearly-bounded heuristic functions can indeed be depicted graphically as a surface above the a - b plane. The theorems of Sections 3.1, 3.2.1, 3.2.2, and 3.2.3 and certain subsequent theorems include the class of linearly bounded heuristic functions as a special case. We shall present the theorems in decreasing level of generality. These theorems are then specialized to the class of linearly-bounded heuristic functions in Section 3.2.4. Pohl [1975] analyzed a special case of linearly-bounded functions.

Returning now to the general case, note that two K functions can have the same characteristic $KMIN$ and $KMAX$ estimate-bounding functions, yet compute different values at any particular node. For example, the K function shown in Figure 3.1-8 is distinct from that in Figure 3.1-5, yet its $\langle KMIN, KMAX \rangle$ are identical to those of the latter. This means that the set of all choices of $\langle KMIN, KMAX \rangle$ partitions the set of all

K functions: two K functions are equivalent iff their corresponding KMIN and KMAX functions are identical. We have thus blurred the distinction between all K functions that happen to have a particular KMIN and KMAX as bounding functions. We can't predict their performances individually, but can only give the best case or average case or worst case performance. The following definition measures performance by the number of nodes expanded in the worst case, as a function of the KMIN and KMAX functions.

Definition 3.1-3. Let $\langle KMIN, KMAX \rangle \in KB^*$. Then $XWORST(M, KMIN, KMAX, W, N)$ denotes the number of nodes of $T(M, N)$ that are expanded during A^* search using evaluation function $F(s) = (1-W) \cdot g(s) + W \cdot KWORST(KMIN, KMAX, s)$, where

$$KWORST(KMIN, KMAX, s) = \begin{cases} KMAX(h(s)) & \text{if } s \text{ is SP} \\ KMIN(h(s)) & \text{if } s \text{ is NSP} \end{cases}$$

Hence $XWORST$ is a particular function of the form

$$\mathbb{N}^+ \times (\mathbb{N} \rightarrow \mathbb{R}^+) \times (\mathbb{N} \rightarrow \mathbb{R}^+) \times [0,1] \times \mathbb{N} \rightarrow \mathbb{N}$$

where $\mathbb{N}^+$ denotes the positive integers.

Figure 3.1-9 shows some of the values of the $KWORST$ function corresponding to the K function of Figure 3.1-5. The fact that the definition of $KWORST$ distinguishes the two cases "s is SP" and "s is NSP" does not require an assumption that a heuristic function used in practice can distinguish SP from NSP nodes. Rather, the definition of $KWORST$ simply defines a mapping from the set KS to itself: for any function $K \in KS$ there exists its corresponding $KWORST$ function also in KS , and it is the number of nodes expanded by A^* using $KWORST$ that we intend to count.

It is easy to see that the number of nodes expanded under the conditions of Theorem 3.1-1 using any $K \in KS$ is bounded above by the $XWORST$ formula, evaluated at the KMIN and KMAX functions characteristic of that K. This follows because the value of the left hand side of (3.1-2) using $KWORST$ never exceeds the value of the left hand side of (3.1-2) using K, and the value of the right hand side of (3.1-2) using K never exceeds that of the right hand side of (3.1-2) using $KWORST$. Hence any node expanded by any K is also expanded when its corresponding $KWORST$ is used instead to guide the search. The inequality (3.1-2) can now be rewritten as

$$\max_{i=p(s)+1}^{g(s)} (1-W) \cdot i + W \cdot KMIN(N + i - 2 \cdot p(s)) \leq \max_{j=p(s)+1}^N (1-W) \cdot j + W \cdot KMAX(N - j)$$

(3.1-4)

The inequality (3.1-4) is expressed in terms of $g(s)$ and $p(s)$, which measure distance from the root node, but it turns out that the analysis is simpler if (3.1-4) is rewritten equivalently in terms of $r(s)$ and $y(s)$, which measure distance from the goal node to the node of divergence of s and distance from node s to its node of divergence, respectively (see Figure 3.1-1). Subtracting from each side of (3.1-4) the quantity $(1-W) \cdot p(s)$ (i.e., the term contributed to each side by the path from the root to the node of divergence) and performing the changes of variable $k = i - p(s)$ and $m = j - p(s)$ and then $y(s) = g(s) - p(s)$ and $r(s) = N - p(s)$ yields the following algebraically equivalent condition:

$$\max_{k=1}^{y(s)} (1-W) \cdot k + W \cdot KMIN(r(s) + k) \leq \max_{m=1}^{r(s)} (1-W) \cdot m + W \cdot KMAX(r(s) - m) \quad (3.1-5)$$

3.1.5 Bounds on Heuristic Distance Estimates Imply Bounds on Lengths of "Garden Paths": The $YMAX(KMIN, KMAX, W, r)$ Function

For any choice of $KMIN$ and $KMAX$ functions, it is clear that each value of $r(s)$ determines a maximum value of $y(s)$ for which (3.1-5) is true. Intuitively, this maximum value of $y(s)$ is the maximum length of a "garden path" originating at the node on the solution path at distance $r(s)$ from the goal node. Formula (3.1-5) says that a NSP node s will be expanded iff $y(s)$ is less than the garden path length corresponding to the node of divergence of s . (In general, the garden paths originating at different nodes on the solution path will vary in length.) This motivates the following definition.

Definition 3.1-4a.

Let $Q1(KMIN, W, i, r)$ denote $(1-W) \cdot i + W \cdot KMIN(r + i)$.

Let $Q2(KMAX, W, i, r)$ denote $(1-W) \cdot i + W \cdot KMAX(r - i)$.

For any positive integer r , let $YMAX(KMIN, KMAX, W, r)$ denote the largest integer k such that for all integers $1 \leq y \leq k$ the following is true.

$$\max_{i=1}^y Q1(KMIN, W, i, r) \leq \max_{j=1}^r Q2(KMAX, W, j, r) \quad (3.1-6)$$

For certain values of the parameters, the left hand side of (3.1-6) exceeds the right hand side for all values of $y \geq 1$. In such cases, $YMAX$ is defined to be zero.

The following equivalent procedural definition of the values of YMAX makes more apparent the relation between the values of KMIN and KMAX and the corresponding values of YMAX. This procedural definition is used in Section 3.5 to calculate YMAX values numerically.

Definition 3.1-4b.

```
integer array YMAX(integer array kmin, kmax; real w; integer r);
begin integer j, k;
real maxq2, tmp;
```

```
real array Q1(integer array kmin; real w; integer i,r);
return((1-w)*i + w*kmin(r+i));
```

```
real array Q2(integer array kmax; real w; integer i,r);
return((1-w)*i + w*kmax(r-i));
```

```
maxq2 ← 0;
for j ← 1 step 1 until r do if (tmp ← Q2(kmax, w, j, r)) > maxq2 then maxq2 ← tmp;
k ← 1;
while Q1(kmin, w, k, r) ≤ maxq2 do k ← k + 1;
return(k-1)
end;
```

Applying the preceding definitions under the conditions of Theorem 3.1-1, using (3.1-5) instead of (3.1-2), Theorem 3.1-1 can be restated to say that a NSP node s is expanded iff

$$y(s) \leq YMAX(KMIN, KMAX, W, r(s)) \quad (3.1-7)$$

Figure 3.1-2 illustrates the relation between the values of YMAX and the number of nodes expanded. Note that YMAX is independent of M and N , except that $0 \leq r(s) \leq N$.

3.1.6 A General Case Theorem: How Many Nodes are Expanded?

The number of nodes expanded for arbitrary values of M , N , $KMIN(i)$, $KMAX(i)$, and W may be expressed directly in terms of the values of YMAX, as follows.

Theorem 3.1-2.

For any $\langle KMIN, KMAX \rangle \in KB^*$, and any $0 \leq W \leq 1$, and any positive integers M and N ,

$$XWORST(M, KMIN, KMAX, W, N) = \sum_{1 \leq i \leq N} M \cdot YMAX(KMIN, KMAX, W, i)$$

Proof. For each $i = 0, 1, 2, \dots, N-1$, the number of sons of u_i that are expanded is either one (namely u_{i+1} , in the case $YMAX(N-i) = 0$), or M , one of which is the node u_{i+1} and the remaining $M-1$ of which are the roots of uniform subtrees of width M and depth $YMAX(KMIN, KMAX, W, N-i) - 1$, in which all nodes are expanded. Let the latter sons of u_i be called the "non-solution-path" sons of u_i , or simply "NSP-sons". Then the total number of nodes in $T(M, N)$ that are expanded is N , the number of nodes on the solution path, plus the sum over i of the number of nodes in the subtrees rooted at the $M-1$ NSP-sons of u_i . The number of nodes in a uniform tree of width M and depth k is

$$\begin{aligned} Z(M, k) &= \sum_{0 \leq i \leq k} M^i \\ &= (M^{k+1} - 1) / (M - 1) \end{aligned}$$

Note that formally $Z(M, -1) = 0$, which is used below for the case $YMAX(KMIN, KMAX, W, i) = 0$. Then

$$\begin{aligned} XWORST(M, KMIN, KMAX, W, N) &= N + \sum_{1 \leq i \leq N} (M-1) \cdot Z(M, YMAX(KMIN, KMAX, W, i) - 1) \\ &= N + \sum_{1 \leq i \leq N} (M-1) \cdot (M^{YMAX(KMIN, KMAX, W, i)} - 1) / (M-1) \\ &= \sum_{1 \leq i \leq N} M^{YMAX(KMIN, KMAX, W, i)} \end{aligned} \quad \square$$

There remains to determine $YMAX(KMIN, KMAX, W, i)$, given $KMIN$ and $KMAX$, using Definition 3.1-4. Inspection of relation (3.1-6) in Definition 3.1-4, however, does not immediately suggest a general closed form expression for $YMAX$. In the face of this apparent obstacle, we make simplifying assumptions for the analysis of the next three sections. In Section 3.5, we apply the results of this section to compute the values of $YMAX(r)$ numerically when the $KMIN$ and $KMAX$ functions are given by lists of numeric values rather than by symbolic formulas, or when the $KMIN$ and $KMAX$ functions fail to satisfy the conditions of the simpler formulas that follow.

As a brief digression, we now compare the present analysis with analyses of other algorithms. We have formulated A^* as a parameterized family of algorithms, in essentially the same way as are the Quicksort-taking-median-of- k -elements algorithm [Sedgewick 1975], the so-called epsilon

approximation algorithms [Garey & Johnson 1976], [Karp 1976], and others. In each of these cases a set of algorithms is defined, as well as an operation mapping each algorithm in the set to its corresponding cost function (whose domain represents the size of the problem and whose range represents the number of steps executed). The principal difference between A^* and the other algorithms mentioned is that for the other families there is a single scalar-valued parameter, whereas A^* is parameterized in the DEBET model by two functions from the non-negative integers to the non-negative reals. (For simplicity, we ignore W here.) One implication of this difference is that there is no obvious linear ordering on the A^* algorithm set as is the case for algorithm sets parameterized by a single scalar.

3.2 Simplifying Assumptions for Analysis

3.2.1 Definitions and Lemmas

So far we have reduced the derivation of a general cost formula having functions as arguments to the marginally less difficult task of deriving a formula for $YMAX$, given symbolic formulas for $KMIN$ and $KMAX$. Our approach now is to assume certain properties of $KMIN$ and $KMAX$ that simplify (3.1-6) and hence permit easier analysis. Specifically, we assume that the F values of NSP nodes increase monotonically with distance from the goal, and that the F values of SP nodes either increase monotonically with distance from the goal or decrease monotonically with distance from the goal. This implies that the sequence described on each side of (3.1-6) takes on its maximum value at one of the extreme elements of the sequence, at $i = y$ for the left hand side (the case of Lemma 3.2-1, below) and at either $j = 1$ or $j = r$ for the right hand side (Lemmas 3.2-2 and 3.2-3, respectively), as depicted in Figure 3.2-1. (We will address shortly how realistic these assumptions are.)

Lemma 3.2-1. If $KMIN < KB$ and $W > 0$ and $KMIN(i+1) \geq KMIN(i) - (1-W)/W$ for all $i = 0, 1, \dots$, then for all $y = 1, 2, \dots$ and $r = 0, 1, \dots$,

$$\max_{i=1}^y Q1(i,r) = Q1(y,r).$$

Proof. It suffices to show for all positive y and non-negative r that $Q(y+1,r) \geq Q(y,r)$. By assumption, for any such y and r ,

$$KMIN(y+r+1) \geq KMIN(y+r) - (1-W)/W$$

Then

$$W \cdot KMIN(y+r+1) \geq W \cdot KMIN(y+r) - (1-W)$$

$$(1-W) \cdot (y+1) + W \cdot KMIN(y+r+1) \geq (1-W) \cdot (y+1) + W \cdot KMIN(y+r) - (1-W)$$

$$= (1-W) \cdot y + W \cdot KMIN(y+r)$$

$$Q1(y+1,r) \geq Q1(y,r) \quad \square$$

In the following two lemmas the symbols i , y , and r have as domains those given in Lemma 3.2-1.

Lemma 3.2-2. If $KMAX < KB$ and $W > 0$ and $KMAX(i+1) \geq KMAX(i) + (1-W)/W$ for all i , then for all r ,

$$\max_{i=1}^r Q2(i,r) = Q2(1,r)$$

Proof. Analogous to proof of Lemma 3.2-1. □

Lemma 3.2-3. If $KMAX < KB$ and $W > 0$ and $KMAX(i+1) \leq KMAX(i) + (1-W)/W$ for all i , then for all r ,

$$\max_{i=1}^r Q2(i,r) = Q2(r,r)$$

Proof. Analogous to proof of Lemma 3.2-1. □

Note that a given KMIN function may satisfy the conditions of Lemma 3.2-1 (or Lemma 3.2-2) for some values of W but not for others. The same applies to KMAX functions with respect to Lemma 3.2-3. For brevity we say that a KMIN function is "IM" (for "F values Increasing Monotonic with distance from goal") if it satisfies the conditions of Lemma 3.2-1. Similarly, we say that a KMAX function is "IM" if it satisfies the conditions of Lemma 3.2-2, or is "DM" if it satisfies Lemma 3.2-3. Given a value for W and a particular $\langle KMIN, KMAX \rangle$ such that KMIN is IM, then we say that $\langle KMIN, KMAX \rangle$ is DM if KMAX is DM, and that $\langle KMIN, KMAX \rangle$ is IM if KMAX is IM.

N.B. When we say $\langle KMIN, KMAX \rangle$ is IM (or DM), we must also identify, either explicitly or implicitly, the value or values of

W for which this is to hold. When not otherwise specified, $W = .5$ is assumed.

Although many $\langle KMIN, KMAX \rangle$ fail to be IM or DM, at least some heuristics used in practice have $KMIN$ and $KMAX$ functions that do satisfy empirically these conditions. In particular, note that the $\langle KMIN, KMAX \rangle$ shown in Figures 2.4-2 and 2.4-3, corresponding to two 8-puzzle heuristics, are DM for $W \leq .5$, suggesting that the IM and DM conditions are not unrealistic. These two $\langle KMIN, KMAX \rangle$ fail to be IM or DM for $W > .5$ because neither $KMIN$ is IM in this range of W . The $\langle KMIN, KMAX \rangle$ of Figure 2.4-4 is DM for $W \leq .09$, but not for larger values of W because $KMAX(4) = 22$ and $KMAX(5) = 32$. This $\langle KMIN, KMAX \rangle$ is not IM for any value of W : although $KMIN$ is IM for all values of W , $KMAX$ is not IM for any value of W because it sometimes decreases with i (e.g., $KMAX(11) = 56$ and $KMAX(12) = 55$). The results of Section 3.1 apply of course to the $\langle KMIN, KMAX \rangle$ of Figure 2.4-2, 2.4-3, and 2.4-4 as to all $\langle KMIN, KMAX \rangle$ for all values of W , so that even for cases in which the simpler formulas of this and the following two sections do not apply, quantitative predictions can still be obtained by means of a numeric computation, as in Section 3.5.

3.2.2 A Theorem Simplifying the Computation of $YMAX(KMIN, KMAX, W, r)$

Applying Theorem 3.1-1, it follows that if $\langle KMIN, KMAX \rangle$ is IM, then a NSP node s is expanded iff $F(s) \leq F(u_{p(s)+1})$. In this case, whenever a SP node is expanded, the possibility for expanding any other extant open node is immediately and permanently eliminated, a sort of "irreversible progress" property. (See Figures 3.1-1 and 3.2-1.) Similarly, if $\langle KMIN, KMAX \rangle$ is DM, then a NSP node s is expanded iff $F(s) \leq F(u_N)$. To assess the extent to which the assumptions described in Lemmas 3.2-1, 3.2-2, and 3.2-3 simplify the analysis, the reader may compare the inequalities given in this paragraph with those given by (3.1-2), (3.1-4) and (3.1-5) in Section 3.1.

Theorem 3.2-4. Given a value for W such that $KMIN$ is IM (or DM) then $YMAX(KMIN, KMAX, W, r)$ is the largest non-negative integer k such that for all non-negative integers $y \leq k$ the following is true.

$$y \leq 1 + W/(1-W) \cdot (KMAX(r-1) - KMIN(r+y)) \quad \text{if } KMAX \text{ is IM and } W < 1 \quad (3.2-1a)$$

$$KMIN(y + r) \leq KMAX(r - 1) \quad \text{if } KMAX \text{ is IM and } W = 1 \quad (3.2-1b)$$

$$y \leq r + W/(1-W) \cdot (KMAX(0) - KMIN(r+y)) \quad \text{if } KMAX \text{ is DM and } W < 1 \quad (3.2-2a)$$

$$KMIN(y+r) \leq r - KMAX(0) \quad \text{if } KMAX \text{ is DM and } W = 1 \quad (3.2-2b)$$

Proof. (KMAX is IM): Applying Lemmas 3.2-1 and 3.2-2 to condition (3.1-4) of Definition 3.1-4a obtains the equivalent condition $Q1(y,r) \leq Q2(1,r)$. Substituting in the latter the definitions of $Q1$ and $Q2$ (see Definition 3.1-4a) obtains:

$$(1-W) \cdot y + W \cdot KMIN(y+r) \leq 1-W + W \cdot KMAX(r-1)$$

implying the following:

$$W < 1: \quad y \leq 1 + W/(1-W) \cdot (KMAX(r-1) - KMIN(r+y))$$

$$W = 1: \quad KMIN(y+r) \leq KMAX(r-1)$$

(KMAX is DM): Analogous, applying Lemmas 3.2-1 and 3.2-3. □

Figure 3.2-2 depicts a geometric interpretation of the computation of $YMAX(r)$ for fixed r by (3.2-1a).

3.2.3 Monotonicity Theorems: Comparing Two Heuristic Functions

Theorem 3.2-4 permits us to prove easily that worst case cost in the case of IM heuristic functions is related monotonically to the difference between $KMAX(i)$ and $KMIN(i)$, as follows.

Theorem 3.2-5. Let $KMAX(i)$, $KMIN_1(i)$, $KMIN_2(i)$, and W be given such that $0 \leq W < 1$, $\langle KMIN_1, KMAX \rangle$ is IM, $\langle KMIN_2, KMAX \rangle$ is IM, and for every $i = 0, 1, 2, \dots$, $KMIN_1(i) \leq KMIN_2(i)$. Then for all $N = 0, 1, 2, \dots$, and $M = 1, 2, \dots$, and $i = 0, 1, 2, \dots$,

$$YMAX(KMIN_2, KMAX, W, i) \leq YMAX(KMIN_1, KMAX, W, i)$$

$$XWORST(M, KMIN_2, KMAX, W, N) \leq XWORST(M, KMIN_1, KMAX, W, N)$$

Proof. By Theorem 3.1-2, the $XWORST$ inequality follows from the $YMAX$ inequality, so it suffices to prove the latter. Applying Theorem 3.2-4, we know by assumption for any y and i that if $y \leq 1 + W(1-W) \cdot (KMAX(i-1) - KMIN_2(i+y))$ then also

$y \leq 1 + W(1-W) \cdot (KMAX(i-1) - KMIN_1(i+y))$. The desired $YMAX$ relation follows. □

Theorem 3.2-5 relates two K functions having identical KMAX(i) functions and different KMIN(i) functions. The following theorem similarly relates two K functions having identical KMIN functions and different KMAX functions.

Theorem 3.2-6. Let $KMIN(i)$, $KMAX_1(i)$, $KMAX_2(i)$, and W be given such that $0 \leq W < 1$, $\langle KMIN, KMAX_1 \rangle$ is IM, $\langle KMIN, KMAX_2 \rangle$ is IM, and for every $i = 0, 1, 2, \dots$, $KMAX_1(i) \leq KMAX_2(i)$. Then for all $N = 0, 1, 2, \dots$, and $M = 1, 2, \dots$, and $i = 0, 1, 2, \dots$,

$$YMAX(KMIN, KMAX_1, W, i) \leq YMAX(KMIN, KMAX_2, W, i)$$

$$XWORST(M, KMIN, KMAX_1, W, N) \leq XWORST(M, KMIN, KMAX_2, W, N)$$

Proof. Analogous to proof of Theorem 3.2-5. □

Combining Theorems 3.2-5 and 3.2-6 we obtain:

Corollary 3.2-7 Given W such that $0 \leq W < 1$, let $\langle KMIN_1, KMAX_1 \rangle$ be IM and $\langle KMIN_2, KMAX_2 \rangle$ be IM such that for every $i = 0, 1, 2, \dots$ $KMIN_1(i) \leq KMIN_2(i)$ and $KMAX_1(i) \geq KMAX_2(i)$. Then for all $0 \leq W < 1$, $N = 0, 1, 2, \dots$, and $M = 1, 2, \dots$,

$$YMAX(KMIN_1, KMAX_1, W, i) \geq YMAX(KMIN_2, KMAX_2, W, i)$$

$$XWORST(M, KMIN_1, KMAX_1, W, N) \geq XWORST(M, KMIN_2, KMAX_2, W, N)$$

Proof. By Theorems 3.2-5 and 3.2-6 and transitivity of " $\leq$ ". □

3.2.4 Application to the Class of "Linearly-Bounded" Heuristic Functions

3.2.4.1 Simple Formulas and Their Geometric Interpretations

As an example in depth, we consider now a class of "linearly-bounded" heuristic functions, for which KMIN and KMAX are linear functions, i.e.,

$$KMIN(i) = a \cdot i$$

$$KMAX(i) = b \cdot i \quad a, b \text{ reals; } 0 \leq a \leq b$$

as illustrated in Figure 3.1-6. Within this class, a heuristic function is identified by an ordered pair of real scalars a and b , so for brevity $\langle a, b \rangle$ denotes the $\langle KMIN, KMAX \rangle$ such that $KMIN(i) = a \cdot i$ and $KMAX(i) = b \cdot i$ for all $i = 0, 1, 2, \dots$. Without loss of generality, we can restrict W to have one of two values, either $W = 1.0$ or any value such that $0 < W < 1$. This follows from two observations: that changing the value of W is equivalent to not changing W and multiplying the heuristic by a scalar, and that the $\langle a, b \rangle$ class is closed under multiplication by a non-negative scalar. The first holds because it is the relative order and not the absolute order of F values that determines which nodes are expanded. Hence the functions $F(s) = .5 \cdot g(s) + .5 \cdot K(s)$ and $F(s) = g(s) + K(s)$ produce identical searches. The functions $F(s) = g(s) + v \cdot K(s)$ and $F(s) = (1 - W) \cdot g(s) + W \cdot K(s)$ are similarly equivalent if the ratios of the weights given to the $g(s)$ term and to the $K(s)$ term are identical, i.e., if $v / 1 = W / (1 - W)$. The second observation states that $\langle a, b \rangle \cdot v = \langle va, vb \rangle$, or in long form, if $K(s)$ has bounding functions $KMIN(i) = a \cdot i$ and $KMAX(i) = b \cdot i$, then $K'(s) = v \cdot K(s)$ has bounding functions $KMIN'(i) = v \cdot a \cdot i$ and $KMAX'(i) = v \cdot b \cdot i$. Hence $YMAX(a, b, W, r) = YMAX(va, vb, .5, r)$, where $v = W / (1 - W)$ and $W < 1.0$. In this section we assume $W = .5$. The case $W = 1.0$ is considered in Section 3.4.

To determine $YMAX(a, b, r)$ by (3.2-1) or (3.2-2), first we establish that $KMIN(i) = a \cdot i$ is IM and that $KMAX(i) = b \cdot i$ is either IM or DM, thus:

$$\begin{aligned} KMIN(i+1) &= a \cdot (i+1) \\ &= KMIN(i) + a \\ &\geq KMIN(i) - 1 \end{aligned}$$

$$\begin{aligned} KMAX(i+1) &= b \cdot (i+1) \\ &= KMAX(i) + b \end{aligned} \quad \begin{array}{l} \text{satisfies IM for } b \geq 1, \\ \text{satisfies DM for } b \leq 1 \end{array}$$

Applying Theorem 3.2-4, $YMAX(a, b, r)$ is determined by the condition:

$$\begin{aligned} b \geq 1: \quad y &\leq 1 + KMAX(r-1) - KMIN(r+y) \\ &= 1 + b \cdot (r-1) - a \cdot (r+y) \\ y &\leq (b-a)/(1+a) \cdot r - (b-1)/(1+a) \end{aligned} \tag{3.2-3}$$

$$\begin{aligned} b \leq 1: \quad y &\leq r - KMIN(r+y) \\ &= r - a \cdot (r+y) \\ y &\leq (1-a)/(1+a) \cdot r \end{aligned} \tag{3.2-4}$$

Let $C1(a, b) = (b-a)/(1+a)$, $C2(a, b) = (1-a)/(1+a)$, $C3(a, b) = (b-1)/(1+a)$

$$\text{Then } YMAX(a, b, r) = \begin{cases} \lfloor C1(a, b) \cdot r - C3(a, b) \rfloor & \text{if } b \geq 1 \\ \lfloor C2(a, b) \cdot r \rfloor & \text{if } b \leq 1 \end{cases} \quad (3.2-5)$$

Note that $YMAX(a, b, r)$ is linear in r , meaning that the maximum distance off the solution path that search can wander (from SP node u_{N-r}) is a fixed fraction (minus a constant in the case $b \geq 1$) of the distance remaining to the goal (i.e, this distance is r , and the fraction is independent of r). See Figure 3.2-3.

$$XWORST(M, a, b, N) = \begin{cases} \sum_{1 \leq i \leq N} M^i \lfloor C1(a, b) \cdot i - C3(a, b) \rfloor & \text{if } b \geq 1 \\ \sum_{1 \leq i \leq N} M^i \lfloor C2(a, b) \cdot i \rfloor & \text{if } b \leq 1 \end{cases}$$

Note that $XWORST(M, a, b, N)$ is exponential in N unless $C1(a, b) = 0$ (when $a=b \geq 1$) or $C2(a, b) = 0$ (when $a=b=1$), in which case $XWORST(M, a, b, N) = N$, which is optimal. Hence search efficiency can be measured by a scalar quantity, the coefficient of the exponent.

$$C(a, b) = \begin{cases} C1(a, b) & \text{if } b \geq 1 \\ C2(a, b) & \text{if } b \leq 1 \end{cases}$$

$$XWORST(M, a, b, N) = \begin{cases} O(M^{\lfloor C(a, b) \cdot N \rfloor}) & \text{if } C(a, b) > 0 \\ N & \text{if } C(a, b) = 0 \end{cases}$$

The cost function $C(a, b)$ defines a surface in 3-space, measured by elevation above the a - b plane. Figure 3.2-4 shows some iso-cost contours of this surface. Note that optimal search performance (i.e., $C(a, b) = 0$) occurs for any $\langle a, b \rangle$ such that $a = b \geq 1$. These $\langle a, b \rangle$ correspond to the heuristic functions $K(s) = ah(s)$, i.e., the exact distance function multiplied by a scalar value greater than 1. But note that $\langle a, b \rangle$ such that $a = b < 1$ has exponential cost. Note also that $C(a, b)$ can be greater than one, meaning that some $\langle a, b \rangle$ expand nodes at levels deeper than N , as in Figure 6b. Such $\langle a, b \rangle$ have worst case performance worse than that of breadth-first search (for which $C(a, b) = 1$). In fact, $C(a, b)$ is unbounded from above.

3.2.4.2 A Scalar Optimization Operation on Linearly-Bounded Heuristic Functions

The concept of improving heuristics can be expressed here in terms of operations on heuristic functions that map each $\langle KMIN, KMAX \rangle$ into another $\langle KMIN, KMAX \rangle$ having perhaps lower cost. Mathematically, we speak of autojections on the set of all $\langle KMIN, KMAX \rangle$. A hypothetical cost-reducing autojection on the $\langle a, b \rangle$ plane is suggested by the arrow in Figure 3.2-5, which shows cuts of the $C(a, b)$ surface. An autojection U is realizable in practice if every instance of $K(s)$ is replaced by $U(K(s))$, where U is a computable function that takes as input the value computed by $K(s)$. The autojection suggested by Figure 3.2-5 (i.e., map $\langle a, b \rangle$ to $\langle b, b \rangle$) is unrealizable in this sense; however, multiplying any $K(s)$ by 3 or 7.5 or by any constant is certainly realizable. The dashed line in Figure 3.2-4 illustrates that the effect of multiplying an arbitrary $\langle a, b \rangle$ by a non-negative scalar v is to map each $\langle a, b \rangle$ into $\langle va, vb \rangle$, corresponding to moving in the a - b plane along the ray through the origin and point (a, b) . This ray crosses $C(a, b)$ contour lines, meaning a change in the coefficient of the exponent in the formula for $XWORST$. Figure 3.2-6 depicts this fact by cuts of the $C(a, b)$ surface along these rays. The value of v that minimizes cost can be determined by inspection in Figure 3.2-6 and is determined formally by taking the derivative of $C(va, vb)$ with respect to v , thus:

$$\begin{aligned} v \cdot b \geq 1: \quad d C(va, vb) / d v &= d ((vb - va) / (1 + va)) / d v \\ &= (b - a) / (1 + va) > 0 \end{aligned}$$

$$\begin{aligned} v \cdot b \leq 1: \quad d C(va, vb) / d v &= d ((1 - va) / (1 + va)) / d v \\ &= -2a / (1 + va) < 0 \end{aligned}$$

Hence for any $\langle a, b \rangle$ such that $b > 0$, the value of v that minimizes cost is $v = 1/b$, transforming $\langle a, b \rangle$ into $\langle a/b, 1 \rangle$, which has cost $C(a/b, 1) = (b-a)/(b+a)$. Graphically interpreted, this scalar optimization maps the $\langle a, b \rangle$ plane into the line segment $b = 1$, $0 \leq a \leq 1$. The optimization scales the heuristic so that its $KMAX(i) = 1$, as illustrated by Figure 3.2-8.

At this point in the analysis, determining the number of nodes expanded $XWORST(M, KMIN, KMAX, W, N)$ requires determining the values of $YMAX(KMIN, KMAX, W, r)$ for each $0 \leq r \leq N$; determining each of the latter requires solving what in general may be a transcendental equation, as was illustrated in Figure

3.2-2. The simplifications of this section become more intuitively meaningful after the following reformulation.

3.3 Cost as a Function of Relative Error in Heuristic Estimates

3.3.1 Definitions

In contrast with the complicated and intuitively unmeaningful formulas of the preceding sections, this section demonstrates that a simple formula for XWORST does exist, if heuristics are expressed in terms of the relative error in their estimates of distance to the goal. (Pohl [1975] obtained results for heuristics expressed in terms of relative error, but only in a restricted case, namely a subset of the set of linearly-bounded heuristic functions of the preceding section.) Our approach is to define functions $\alpha(i)$ and $\delta(i)$ such that the bounding functions $KMIN(i)$ and $KMAX(i)$ can in general be rewritten as

$$KMIN(i) = (1 - \delta(i)) \cdot \alpha(i) \cdot i \quad (3.3-1)$$

$$KMAX(i) = (1 + \delta(i)) \cdot \alpha(i) \cdot i$$

Solving for $\delta(i)$ and $\alpha(i)$ in terms of $KMIN(i)$ and $KMAX(i)$, we obtain:

Definition 3.3-1. Let $\delta(KMIN, KMAX, i) \equiv 0$ if $KMAX(i) = 0$. Let $\alpha(KMIN, KMAX, 0) \equiv 0$. Otherwise for $i = 0, 1, 2, \dots$ let

$$\delta(KMIN, KMAX, i) \equiv (KMAX(i) - KMIN(i)) / (KMAX(i) + KMIN(i))$$

$$\alpha(KMIN, KMAX, i) \equiv (KMIN(i) + KMAX(i)) / 2 \cdot i$$

Hence any $\langle KMIN, KMAX \rangle$ function can be expressed in its equivalent $\langle \alpha, \delta \rangle$ form, and vice versa. Figures 3.3-1, 3.3-2, and 3.3-3 illustrate the relation between $KMIN(i)$ and $KMAX(i)$ functions and the corresponding $\delta(i)$ and $\alpha(i)$ functions: Figure 3.3-1 for the $\langle a, b \rangle$ heuristic shown in Figure 3.1-6, Figure 3.3-2 for an arbitrary

$\langle KMIN, KMAX \rangle$, and Figure 3.3-3 for the $\langle KMIN, KMAX \rangle$ corresponding to the 8-puzzle heuristic K_2 defined in Chapter 2. For the case of linearly-bounded heuristic functions, we have that

$$\delta(a, b, i) = (b \cdot i - a \cdot i) / (b \cdot i + a \cdot i) = (b - a) / (b + a)$$

$$\alpha(a, b, i) = (a + b) / 2$$

So for any particular $\langle a, b \rangle$, $\delta(a, b, i)$ is independent of i (meaning that the maximum relative error in $\langle a, b \rangle$ remains constant with distance from the goal), so we write simply $\delta(a, b)$. (Incidentally, note that $\delta(a, b) = C(a/b, 1)$, because multiplying a $\langle KMIN, KMAX \rangle$ by a scalar leaves its δ invariant.) If $KMIN(i) \leq K(s) \leq KMAX(i)$ for all s such that $h(s) = i$ and for all i , then $|K(s) - \alpha(i)| / \alpha(i) \leq \delta(i)$. For this reason we refer to $\delta(i)$ as the "maximum relative error function of K " or the "maximum relative error function of $\langle KMIN, KMAX \rangle$ ". Note that Definition 3.3-1 implies that $0 \leq \delta(KMIN, KMAX, i) \leq 1$ for all $KMIN, KMAX$, and i .

3.3.2 A Theorem Relating "Garden Path" Length to Relative Error

Theorem 3.3-1 If $KMAX(i) = i$ and if $KMIN(i)$ is IM at $W = .5$ (this condition henceforth abbreviated to " $\langle KMIN, KMAX \rangle$ is IM-never-overestimating"), then $YMAX(KMIN, KMAX, .5, r)$ is the largest non-negative integer k such that for all non-negative integers $y \leq k$

$$y \leq \delta(KMIN, KMAX, r+y) \cdot r$$

Proof. Since $KMAX(i)$ is fixed, $\alpha(i)$ in equation 11.1 can be rewritten in terms of $\delta(i)$, thus:

$$\begin{aligned} \alpha(i) &= (KMIN(i) + i) / 2 \\ &= ((1 - \delta(i)) \cdot \alpha(i) \cdot i + i) / 2 \cdot i \end{aligned}$$

Solving for $\alpha(i)$:

$$\alpha(i) = 1 / (1 + \delta(i))$$

So $KMIN(i)$ in equation (3.3-1) can be rewritten

$$KMIN(i) = i \cdot (1 - \delta(i)) / (1 + \delta(i)) \quad (3.3-2)$$

Substituting the latter in (3.2-1a) with $W = .5$ and letting $\delta(i)$ stand for

$\delta(KMIN, KMAX, i)$, we obtain

$$\begin{aligned} y &\leq 1 + KMAX(r-1) - KMIN(r+y) \\ &= 1 + r - 1 - (r+y) \cdot (1 - \delta(r+y)) / (1 + \delta(r+y)) \end{aligned}$$

Isolating y :

$$y \cdot (1 + (1 - \delta(r+y)) / (1 + \delta(r+y))) \leq r \cdot (1 - (1 - \delta(r+y)) / (1 + \delta(r+y)))$$

$$y \leq \delta(r+y) \cdot r$$

□

This result establishes a close relation between $YMAX$ and relative error. Indeed, for the $\langle a, b \rangle$ class, $\delta(i)$ is a constant function and hence for $\langle a, b \rangle$ heuristics that are IM-never-overestimating (i.e., those for which $b = 1$),

$$YMAX(a, 1, .5, r) = \lfloor \delta(a, 1) \cdot r \rfloor$$

$$XWORST(M, a, 1, .5, N) = \sum_{1 \leq i \leq N} M \cdot \lfloor \delta(a, 1) \cdot i \rfloor$$

In words, the distance to which the search can extend from the solution path is a fraction of the distance from that solution path point to the goal, and this fraction equals the value of the relative error of the heuristic. For arbitrary $\langle a, b \rangle$, the relation between cost and relative error is almost as simple. For $b \geq 1$, formula substitution shows that $\delta(a, b) = C(a, b) / (1 + C3(a, b))$. (Refer to formula (3.2-5).)

Figure 3.3-4 shows a geometric interpretation of the relation between $YMAX(i)$ and $\delta(i)$ values for an arbitrary $\delta(i)$. The heuristic is shown in $\langle KMIN, KMAX \rangle$ form (Figure 3.3-4a) and in $\delta(i)$ form (Figure 3.3-4b), followed by illustrative plots corresponding to the computation of $YMAX(KMIN, KMAX, r)$ for $r = 2$ and $r = 5$. The plot

of $\delta(y+r) \cdot r$ is the plot of $\delta(y)$ shifted left by r units and scaled vertically by a factor of r . The intersection of this curve with the 45 degree line determines the value of $YMAX(r)$, namely if $yint(r)$ is a real-valued quantity denoting the y coordinate of the intersection point, then $YMAX(r) = \lfloor yint(r) \rfloor$ (as in Figure 3.2-2). Figure 3.3-4e compares $yint(r)/r$ with $\delta(r)$.

Figure 3.3-5 shows a heuristic having $\delta(i)$ that is weakly monotonic increasing with i , and also shows its corresponding $yint(r)/r$. For linearly-bounded heuristics, the plot of $\delta(y+r)$ is a horizontal line, hence $yint(r)/r = \delta(r)$.

3.3.3 A Simple Formula Bounding Cost as a Function of Relative Error

The preceding examples serve to motivate the following theorem, which establishes that $YMAX(i)$ is bounded below by $\lfloor \delta(i) \cdot i \rfloor$ if $\delta(i)$ increases monotonically with i , and that $YMAX(i)$ is bounded from above by $\lfloor \delta(i) \cdot i \rfloor$ if δ decreases monotonically with i .

Theorem 3.3-2. Assume $\langle KMIN, KMAX \rangle$ is IM-never-overestimating, and let $\delta(i)$ stand for $\delta(KMIN, KMAX, i)$. Then for any M and for any $N > 0$:

$$YMAX(KMIN, KMAX, .5, i) \leq \lfloor \delta(i) \cdot i \rfloor \quad (3.3-3a)$$

$$XWORST(M, KMIN, KMAX, .5, N) \leq \sum_{1 \leq i \leq N} M \lfloor \delta(i) \cdot i \rfloor \quad (3.3-3b)$$

If $\delta(i)$ is weakly monotonic decreasing for $i > 0$, and

$$YMAX(KMIN, KMAX, .5, i) \geq \lfloor \delta(i) \cdot i \rfloor \quad (3.3-3c)$$

$$XWORST(M, KMIN, KMAX, .5, N) \geq \sum_{1 \leq i \leq N} M \lfloor \delta(i) \cdot i \rfloor \quad (3.3-3d)$$

If $\delta(i)$ is weakly monotonic increasing for $i > 0$.

Proof. Equation (3.3-3b) follows from (3.3-3a) by Theorem 2.1-2, and similarly (3.3-3d) from (3.3-3c). It remains to establish (3.3-3a) and (3.3-3c).

Case (3.3-3c): For any particular i , assume the converse, that $YMAX(i) < \lfloor \delta(i) \cdot i \rfloor$. Hence $YMAX(i) + 1 \leq \lfloor \delta(i) \cdot i \rfloor$, since $YMAX(i)$ has integral value. By Theorem 3.3-1, $YMAX(i) + 1 > i \cdot \delta(YMAX(i) + 1 + i)$. Hence $\lfloor \delta(i) \cdot i \rfloor > i \cdot \delta(YMAX(i) + 1 + i)$, contradicting the assumption that $\delta(i)$ is weakly monotonic increasing.

Case (3.3-3a): For any particular i , assume the converse, that $YMAX(i) > \lfloor \delta(i) \cdot i \rfloor$. By Theorem 3.3-1, $YMAX(i) \leq i \cdot \delta(YMAX(i) + i)$, implying that $YMAX(i) \leq \lfloor i \cdot \delta(YMAX(i) + i) \rfloor$, since $YMAX(i)$ has integral value. Hence $\lfloor \delta(i) \cdot i \rfloor < \lfloor i \cdot \delta(YMAX(i) + i) \rfloor$, contradicting the assumption that $\delta(i)$ is weakly monotonic decreasing. $\square$

Note for the case of linearly-bounded heuristics that $YMAX(a, b, r) = \lfloor \delta(a, b) \cdot r \rfloor$ is implied as a special case of Theorem 3.3-2: the function $\delta(i) = c$, where c is a constant, is both weakly monotonic increasing and weakly monotonic decreasing, hence both (3.3-3b) and (3.3-3d) hold, implying equality.

Using Theorems 3.1-2 and 3.3-2, one can determine δ functions for which XWORST grows at most linearly, polynomially, or exponentially in N . In words, we determine how much relative error in the heuristic distance estimates can be tolerated and still guarantee a cost function that grows within certain bounds. In the following table, Theorem 3.3-2 is used to find $YMAX$ from δ . (c is an arbitrary positive-real-valued constant.) XWORST is determined from $YMAX$ by Theorem 3.1-2, using the properties of the floor function to simplify the expression algebraically. Figure 3.3-6 shows $\langle KMIN, KMAX \rangle$ corresponding to these choices of δ .

$\delta(KMIN, KMAX, r)$	$YMAX(KMIN, KMAX, r)$	$XWORST(M, KMIN, KMAX, N)$	
c	$c \cdot r$	$O(M^{c \cdot N})$	(exponential in N)
$1 / \sqrt{r}$	$\leq \sqrt{r}$	$\leq O(\sqrt{N}) M^{\sqrt{N}}$	(subexponential)
$\log r / r$	$\leq \log(r)$	$\leq O(N^{\log M})$	(polynomial)
c / r	$\leq c$	$\leq N M^c$	(linear)

This table expresses a guarantee: if a K function meets the specified condition, then its performance is no worse than that indicated above. The extremes of this table can be summarized succinctly: constant absolute error (i.e., $\delta(r) = c/r$) gives linear growth in $XWORST$ with N ; constant relative error (i.e., $\delta(r) = c$) gives exponential growth in $XWORST$ with N .

These "limits to growth" results are somewhat sobering: A^* must be given a heuristic function whose absolute accuracy decreases but slightly with distance from the goal node in order to guarantee good performance in the worst case. In contrast, some heuristics used in practice may have the property of estimating most accurately near the goal, with accuracy progressively worsening with increasing distance from the goal. But even relative error that is constant with distance from the goal still causes exponential growth in cost. Determining corresponding results for average case cost remains an open problem.

Of course, the formula in (3.3-3a), on which the entries in the above table are based, gives in general only an upper bound on the values of $YMAX$. Hence this formula alone does not permit us to determine, for two arbitrary given heuristic functions (call them K_u and K_v), whether the $YMAX(i)$ values for K_u dominate those of K_v . For example, suppose that $\delta(i)$ for K_u is never more than $\delta(i)$ for K_v for all $i = 0, 1, 2, \dots$; in this situation we cannot conclude from formula (3.3-3a) that for all $i = 0, 1, 2, \dots$, $YMAX(i)$ for K_u is never more than $YMAX(i)$ for K_v , since a function can be bounded by infinitely many other functions with infinitely many growth rates.

To determine how closely the upper bound on $YMAX(i)$ given by formula (3.3-3a) approximates the exact values of $YMAX(i)$ given by Theorem 3.3-1, we evaluated the

alternative formulas numerically for the several cases tabulated below. In this table $YMAX(i)$ is by Theorem 3.3-1 and $UYMAX(i)$ denotes the upper bound on $YMAX(i)$ given by (3.3-3a).

$\delta(i)$:	$1 / \sqrt{i}$		$\log i / i$	
i	$YMAX(i)$	$UYMAX(i)$	$YMAX(i)$	$UYMAX(i)$
5	2	2	1	1
10	2	3	2	2
15	3	3	2	2
20	3	4	2	2
25	3	5	2	3
30	3	5	2	3
35	4	5	2	3
40	4	6	2	3
45	4	6	2	3
50	4	7	2	3
60	5	7	3	4
70	5	8	3	4
80	5	8	3	4
90	5	9	3	4

Table 3.3-1 Exact values of and upper bounds on $\delta(i)$

Theorem 3.3-1 does not apply to heuristic functions that are not "IM-never-overestimating", but Theorems 3.1-1 and 3.1-2 and Definition 3.1-4 apply to all $\langle KMIN, KMAX \rangle$ functions (Section 3.5 gives an example). Hence given the numeric values of $KMIN(i)$ and $KMAX(i)$ for each of several heuristic functions, we can always compute the exact values of $YMAX$, and hence determine which gives the smallest values of $XWORST$.

3.3.4 Theorems: Cost Grows Monotonically with Relative Error

As an alternative to numerical computation as suggested above, it would be desirable to determine some simple, intuitively meaningful criterion relating the relative costs (measured by $XWORST$) of arbitrary heuristic functions. Above we suggested the possibility that cost of IM-never-overestimating heuristic functions grows monotonically with relative error. We now prove that this is the case.

Theorem 3.3-3. Let K stand for $\langle K_{\min}, K_{\max} \rangle$ and let $XWORST(M, K, N)$ stand for $XWORST(M, K_{\min}, K_{\max}, .5, N)$. Let K_1 and K_2 be IM-never-overestimating. If $\delta(K_1, i) \leq \delta(K_2, i)$ for all i , then for every M and N

$$XWORST(M, K_1, N) \leq XWORST(M, K_2, N)$$

Proof. Invoking Theorem 3.1-2, it suffices to show that $YMAX(K_1, i) \leq YMAX(K_2, i)$ for all i . Applying Theorem 3.3-1, by assumption for any y and i , if $y \leq \delta_1(y+i) \cdot i$ then $y \leq \delta_2(y+i) \cdot i$. The desired $YMAX$ relation follows. $\square$

Alternative proof. By Theorem 3.2-5 it suffices to establish that $KMIN_1(i) \geq KMIN_2(i)$ for all i . By assumption, $KMAX(i) = i = (1 + \delta(i)) \alpha(i) i$. Hence $\alpha(i) = 1 / (1 + \delta(i))$. Substituting,

$$KMIN(i) = (1 - \delta(i)) \alpha(i) i = i (1 - \delta(i)) / (1 + \delta(i))$$

Then $KMIN_1(i) \geq KMIN_2(i)$ iff

$$i (1 - \delta_1(i)) / (1 + \delta_1(i)) \geq i (1 - \delta_2(i)) / (1 + \delta_2(i))$$

Simplifying, we must show that $\delta_2(i) - \delta_1(i) \geq \delta_1(i) - \delta_2(i)$. The latter condition is valid for all i because by assumption $\delta_1(i) \leq \delta_2(i)$ for all i . $\square$

Under the IM-never-overestimating condition assumed in Theorem 3.3-3, if $\delta(K_1, i) \leq \delta(K_2, i)$ then also $\alpha(K_1, i) \geq \alpha(K_2, i)$. A stronger statement about the monotonicity of cost with relative error can be made by comparing $\langle K_{\min}, K_{\max} \rangle$ having identical $\alpha(i)$ and differing $\delta(i)$. The case $\alpha(i) = 1$ is interesting in its own right, however we can prove a much more general result, for arbitrary $\alpha(i)$.

Theorem 3.3-4. Let K stand for $\langle K_{\min}, K_{\max} \rangle$ and let $XWORST(M, K, N)$ stand for $XWORST(M, K_{\min}, K_{\max}, .5, N)$. Let K_1 and K_2 be IM such that $\alpha(K_1, i) = \alpha(K_2, i)$ for all i and $\delta(K_1, i) \leq \delta(K_2, i)$ for all i . Then for every M and N

$$XWORST(M, K_1, N) \leq XWORST(M, K_2, N)$$

Proof. By Theorem 3.1-2, it suffices to show that $YMAX(K_1, i) \leq YMAX(K_2, i)$ for all i . To show this, by Theorem 3.2-7 it suffices to show that $KMIN_1(i) \geq KMIN_2(i)$ for all i

and $KMAX_1 \leq KMAX_2(i)$ for all i . Since $\alpha(K_1, i) = \alpha(K_2, i) = \alpha(i)$, we have $KMIN_1(i) = (1 - \delta_1(i)) \alpha(i) i$ and $KMIN_2(i) = (2 - \delta_2(i)) \alpha(i) i$ and $KMAX_1(i) = (1 - \delta_1(i)) \alpha(i) i$ and $KMAX_2(i) = (2 - \delta_2(i)) \alpha(i) i$. The desired relations follow because $\delta_1(i) \leq \delta_2(i)$ for all i by assumption. $\square$

3.3.5 Lattice Formulation

Theorem 3.3-3 can be restated as follows as a statement about partial orderings, as illustrated in Figure 3.3-7. Definition 3.3-1 implies that for any $KMIN(i)$, $KMAX(i)$ and i , $0 \leq \delta(KMIN(i), KMAX(i), i) \leq 1$. Let D denote the set of all real-valued functions $\delta: \mathbb{N} \rightarrow [0,1]$. Let $\leq_f$ denote the "nowhere greater than" relation between real-valued functions on the non-negative integers: $f_1 \leq_f f_2$ iff $f_1(i) \leq f_2(i)$ for all $i = 0, 1, \dots$. It is easily verified that $\leq_f$ is reflexive, anti-symmetric, and transitive over D , and hence $P = (D, \leq_f)$ is a partial ordering (in fact, a complete infinite continuous lattice [Birkhoff 1963]). If $\delta \in D$, then let $XWORST'(\delta)$ denote the image function under the cost mapping. Clearly, $(\{XWORST'(\delta) \mid \delta \in D\}, \leq_f)$ is also a partial ordering. Theorem 3.3-3 can then be restated thus:

Theorem 3.3-3'. If $\delta_1, \delta_2 \in D$ and $\delta_1 \leq_f \delta_2$ then $XWORST'(\delta_1) \leq_f XWORST'(\delta_2)$.

That is, the mapping from the partial ordering of relative error functions to the partial ordering of cost functions is monotonic. Theorem 3.3-3' says nothing about pairs of K functions that are incomparable under $\leq_f$.

As a trivial example of how general results of lattice theory may be applied to the further analysis of the DEBET model, consider a heuristic function $K_3(s) = \max(K_1(s), K_2(s))$ where K_1 and K_2 are arbitrary such that $\delta_1, \delta_2 \in D$. If $\delta(i) = 0$ is the "bottom" of the lattice P and $\delta(i) = 1$ is the "top", then δ_3 is the "join" of δ_1 and δ_2 , implying $\delta_1 \leq_f \delta_3$ and $\delta_2 \leq_f \delta_3$, and hence $XWORST'(\delta_1) \leq_f XWORST'(\delta_3)$ and

$$XWORST'(\delta_2) \leq_f XWORST'(\delta_3).$$

We have just defined a lattice consisting of all IM-never-overestimating heuristic functions; this class is a subset of the class of all IM/DM heuristic functions. We can extend the lattice formulation to this latter class. Any IM-never-overestimating heuristic function can be specified by a single function, either $KMIN(i)$ (since $KMAX(i) = i$ is assumed), or $\delta(i)$ (hence determining $\omega(i)$), or $\omega(i)$ (hence determining δ). The specification of an IM/DM heuristic function, however, in general requires two functions, either $KMIN(i)$ and $KMAX(i)$ or $\delta(i)$ and $\omega(i)$. Clearly, the "nowhere greater than" relation can be imposed on pairs of functions, as follows. Let $\leq_k$ denote the following relation on $KB^* \times KB^*$: $\langle KMIN_1, KMAX_1 \rangle \leq_k \langle KMIN_2, KMAX_2 \rangle$ iff $KMIN_1(i) \geq KMIN_2(i)$ and $KMAX_1(i) \leq KMAX_2(i)$ for all i . Hence $\leq_k$ is a partial ordering on $KB^* \times KB^*$. Theorems 3.2-5, 3.2-6, and 3.2-7 can then be expressed in lattice form.

See [Ibaraki 1976] for a related study of a partial ordering on a class of branch and bound algorithms generalized to include heuristic search.

3.4 Parameter Tuning: When is Insurance Justified?

3.4.1 Introduction

In practice, performance is sometimes highly sensitive to the choice of relative weights assigned to different terms in an evaluation function. The 8-puzzle is a case in point: simply by increasing W a heuristic function (K_1 or K_2) having a cost function that grows apparently exponentially with N becomes apparently subexponential (Chapter 2). It is therefore of potential practical interest to determine analytically for each heuristic function which value of W is optimal. Previous analyses have focussed on comparing cost for the two values $W = .5$ and $W = 1$, corresponding to F functions of the form $F(s) = g(s) + K(s)$ (call this form A) and $F(s) = K(s)$ (form B). It has been shown [Pohl 1970a] that form A expands fewer nodes than form B for the case

$KMIN(i) = i-a$ and $KMAX(i) = i+b$, but this result lacks generality with respect to $\langle KMIN, KMAX \rangle$, and furthermore says nothing about the case of intermediate values of W .

In this section we first argue intuitively for and against the value of the $g(s)$ term as buying "insurance" against excessive search cost. Then we derive a theorem stating that for arbitrary IM-never-overestimating heuristics, of all $0 \leq W \leq 1$ minimal cost occurs when $W = .5$. Regarding heuristics that are not IM-never-overestimating, we then consider the case of $\langle a, b \rangle$ heuristics, identifying the locus for which form B expands fewer nodes than form A, and by how much, and plotting these results graphically.

If $K(s) = h(s)$, i.e., if $KMIN(i) = KMAX(i) = i$ for all i , then both forms are equivalent and optimal, i.e., $XWORST(N) = N$. For arbitrary $K(s)$ one may argue intuitively that form B has better performance, invoking the maxim "Ignore costs already incurred when deciding what is best to do next". However, if $K(s)$ is very errorful in its estimates, it may cause the search of many garden paths while making minimal progress to the goal. This argues for being conservative by including $g(s)$, the distance from the root, with the effect of insuring that both number and length of garden paths are bounded quantities. So unless $K(s)$ is very accurate, one may argue intuitively that form A is better. How much is "very"? Can't say; this argument is qualitative rather than quantitative. How much "better"? Again, no precise answer is given by this intuitive argument. Similar speculations are given in [Pohl 1970a], [Pohl 1970b], [Nilsson 1971], and [Vanderbrug 1976]. In contrast, DEBET provides exact answers within a restricted context.

3.4.2 Theorem: $W = .5$ is Optimal for "IM-Never-Overestimating" Heuristics

Theorem 3.4-1. If $\langle KMIN, KMAX \rangle$ is IM-never-overestimating, then for all M and N and all $0 \leq W \leq 1$, if $KMIN$ is IM at W (i.e., $KMIN$ satisfies the condition of Lemma 3.2-1) then

$$XWORST(M, KMIN, KMAX, .5, N) \leq XWORST(M, KMIN, KMAX, W, N)$$

Proof. It suffices to show that $YMAX(KMIN, KMAX, .5, r) \leq YMAX(KMIN, KMAX, W, r)$ for all $0 \leq r \leq N$. By Theorem 3.3-2 the value of $YMAX$ for $W = .5$ is determined by the condition $y \leq \delta(y + r) \cdot r$. The proof strategy is to obtain similar conditions for

arbitrary W by application of Theorem 3.2-4, specialized for the case $KMAX(i) = i$, and then to show that for any y and r , $y \leq \delta(y+r)r$ implies that y is less than the expression so obtained. The case $.5 \leq W < 1$ is covered by application of inequality (3.2-1a) in Theorem 3.2-4, the case $W = 1$ by (3.2-1b), and the case $0 \leq W \leq .5$ by (3.2-2a).

Case $.5 \leq W < 1$: $KMAX(i) = i$ is IM for any $.5 \leq W \leq 1$ (see Lemma 3.2-2). If $KMIN$ is IM at W , then inequality (3.2-1a) of Theorem 3.2-4 holds. Following the proof of Theorem 3.3-1 for the case of arbitrary W , and for brevity letting $v = W/(1 - W)$, the following version of (3.2-1a) is obtained:

$$\begin{aligned} y &\leq 1 + v \cdot (r - 1 - KMIN(r+y)) \\ &= 1 + v \cdot (r - 1 - (r + y) \cdot \frac{1 - \delta(r+y)}{1 + \delta(r+y)}) \end{aligned}$$

Isolating y and abbreviating $\delta(r + y)$ to δ :

$$y \leq \frac{2 \cdot v \cdot \delta \cdot r + (1-v) \cdot (1 + \delta)}{1 + \delta + v \cdot (1 - \delta)} \quad (3.3-4)$$

Hence $YMAX(KMIN, KMAX, .5, r) \leq YMAX(KMIN, KMAX, W, r)$ iff $y \leq \delta(y+r) \cdot r$ implies (3.3-4) for all y and r . We postulate this condition and determine when it fails to hold, thus:

$$\delta \cdot r \leq \frac{2 \cdot v \cdot \delta \cdot r + (1-v) \cdot (1 + \delta)}{1 + \delta + v \cdot (1 - \delta)}$$

simplifying to

$$\frac{(\delta \cdot r - 1) \cdot (v - 1) \cdot (1 + \delta)}{1 + \delta + v \cdot (1 - \delta)} \geq 0$$

The terms $v - 1$, $1 + \delta$, and $1 + \delta + v(1 - \delta)$ are never negative, so the latter condition holds iff $r \geq 1/\delta$, or in long form, iff $r \geq 1/\delta(r+y)$. But $r < 1/\delta(r+y)$ implies that $r \cdot \delta(r+y) < 1$, so that $YMAX(KMIN, KMAX, .5, r) = 0$, and hence $YMAX(KMIN, KMAX, .5, r) \leq YMAX(KMIN, KMAX, W, r)$ for all $.5 \leq W < 1$.

Case $W = 1$: In similar fashion, the following version of (3.2-1b) is obtained:

$$(r + y) \cdot (1 - \delta) / (1 + \delta) \leq r - 1$$

Isolating y :

$$y \leq (2 \cdot \delta \cdot r - \delta - 1) / (1 - \delta)$$

So the proof is finished if $\delta \cdot r \leq (2\delta \cdot r - \delta - 1) / (1 - \delta)$

Simplifying, this holds iff $r \geq (1 - \delta) / (\delta \cdot (1 + \delta))$, but if $r < (1 - \delta) / (\delta \cdot (1 + \delta))$ then $r \cdot \delta < (1 - \delta) / (1 + \delta) \leq 1$, implying that in this case $YMAX(KMIN, KMAX, .5, r) = 0$.

Case $0 \leq W \leq .5$: Similar proof, using (3.2-2a). □

Note that this result for the worst case is in disagreement with the experimental data for 8-puzzle search in the average case reported in Chapter 2, for which $W = 1.0$ minimizes cost, at least for large N .

3.4.3 W-Optimality for "Linearly-Bounded" Heuristic Functions

Regarding $\langle KMIN, KMAX \rangle$ that are not IM-never-overestimating, we consider the class of linearly-bounded heuristic functions, for which the preceding theorem applies if $b = 1$. In general, associated with each point $\langle a, b \rangle$ and value of W is a real-valued quantity $C(a, b, W)$. Hence we compare $C(a, b, .5)$ (form A), with $C(a, b, 1.0)$, the corresponding cost function using form B. The function $C(a, b, .5)$ defines a surface in 3-space, as does $C(a, b, 1.0)$, so form A expands fewer nodes than form B for the locus of points $\langle a, b \rangle$ for which the $C(a, b, .5)$ surface lies below the $C(a, b, 1.0)$ surface in elevation above the plane, i.e., the locus of points that satisfy the condition $C(a, b, .5) < C(a, b, 1.0)$.

The locus of $C(a, b, .5) = C(a, b, 1.0)$ is the curve or set of curves (in 3-space) in which the two surfaces intersect. Call this locus the "cost surface intersection locus". The locus of points $\langle a, b \rangle$ in the plane such that $C(a, b, .5) = C(a, b, 1.0)$ is the projection of the cost surface intersection locus onto the (a, b) plane. Call this locus the "intersection projection locus". To determine this locus, there remains only the task of determining the function $C(a, b, 1.0)$. Either by taking the limit of $C(va, vb)$ as v goes to infinity, or by instantiating Theorem 3.2-4 for $W = 1.0$, we obtain $C(a, b, 1.0) = (b-a)/a$. Then the two forms can be compared as follows.

$$b \geq 1: \quad C(a, b, 1.0) = (b-a)/a > (b-a)/(1+a) = C(a, b, .5) \text{ except for } b = a$$

$$b \leq 1: \quad C(a, b, 1.0) = C(a, b, .5) \quad \text{when } (b-a)/a = (1-a)/(1+a)$$

Thus the intersection projection locus consists of those points $\langle a, b \rangle$ satisfying the condition $b = 2a/(1+a)$ for $a < 1$, and satisfying $b = a$ for $a \geq 1$, plotted in Figure 3.4-1. Form A expands fewer nodes than form B for all $b > 2a/(1+a) \geq a$; form B expands the fewer for all $2a/(1+a) > b \geq a$. The difference in C between form A and form B is

$$C(a, b, 1.0) - C(a, b, .5) = \begin{cases} (b-a) / a(1+a) & b \geq 1 \\ (1-ba) / a(1+a) & b \leq 1 \end{cases}$$

The answers supplied by intuitive arguments and by deduction in the DEBET model demonstrate a striking difference in precision. Note that for cases in which analysis is difficult, e.g., in which deriving YMAX may require solving transcendental equations, a numeric computation of the sort described in the following section suffices to answer any particular instantiation of the insurance question.

3.5 Analytic Predictions vs. Experimental Measurements for 8-Puzzle Heuristic Functions

A beautiful theory, killed by a nasty, ugly, little fact.
Thomas Huxley

I speak without exaggeration when I say that I have constructed three thousand theories in connection with the electric light...Yet in only two cases did my experiments prove the truth of my theory.

Thomas Edison

3.5.1 Numerical Comparisons

A model is realistic to the extent that its predictions agree with the experimental measurements that they purport to model. The purpose of this section is to test experimentally the quantitative predictions of the DEBET model using heuristic functions K_1 , K_2 , and K_3 for the 8-puzzle as test cases. Note first the differences between the 8-puzzle and its image in the DEBET model. The 8-puzzle graph is not a

uniform tree, nor do the 8-puzzle heuristics K_1 , K_2 , or K_3 defined in Chapter 2 satisfy the model assumptions about the behavior of the KWORST function. Nevertheless we can instantiate an image within the DEBET model of each of these three 8-puzzle heuristic functions, using the $\langle KMIN, KMAX \rangle$ data in Figures 2.4-2, 2.4-3, and 2.4-4 (in Chapter 2). We can instantiate an image of the 8-puzzle graph in the DEBET model by choosing a suitable value of M . We determine such a value for M experimentally by counting the number of nodes $t(i)$ at each level i of a breadth-first expansion tree of the 8-puzzle graph to depth 14, and fitting an exponential function of the form $t(i) = a \cdot M^i$ to these $(i, t(i))$ values by the least squares method. By this method, we obtained the approximation $M = 1.637$.

Instead of comparing the $XWORST(M, KMAX, KMIN, W, N)$ predictions against the experimental measurements of $XMEAN(K, W, N)$, we compare $XWORST$ values to $XMAX(K, W, N)$ values, which represent the worst case performance observed during the experiments of Chapter 2. Figure 2.3-14 plots the experimentally measured values of $XMAX(K_2, W, N)$ for the 8-puzzle for several values of W . Comparing with Figure 2.3-1, we see that the $XMAX$ values change with W in roughly the same manner as do the corresponding $XMEAN$ values. The cases for K_1 and K_3 are similar but are not plotted here.

The simple formulas given in Sections 3.2 and 3.3 apply for the $\langle KMIN, KMAX \rangle$ values in Figures 2.4-2, 2.4-3, and 2.4-4 for some values of W but not for others. This, together with the fact that these values are more easily given by a list of numeric values than by a symbolic formula, motivate the following numerical computation of $XWORST$. We combine Definition 3.1-4b and Theorem 3.1-2 into an (obvious) algorithm for computing numerically the value of $XWORST(M, KMIN, KMAX, W, N)$ for any value of N , given any particular choice of values for M , $KMIN(i)$, $KMAX(i)$, and W . (The procedure given in Definition 3.1-4b is modified in an obvious way to account for the fact that in practice $KMIN(i)$ and $KMAX(i)$ are known for a limited number of values of i .)

Figure 3.5-1 shows the $XWORST$ values corresponding to the $\langle KMIN, KMAX \rangle$ data for K_2 shown in Figure 2.4-3. Note that $XWORST$ values are given only for $N \leq 17$ (in the case of $W = .5$), whereas $KMIN(i)$ and $KMAX(i)$ values are known for $N \leq 26$. This restriction occurs because for the given $KMIN$ and $KMAX$ values it happens that $YMAX(17) \leq 26 - 17 = 9$ but $YMAX(18) > 26 - 18 = 8$, and hence the value of $YMAX(18)$ cannot be computed since $KMIN$ and $KMAX$ are not known for $i > 26$.

Combining Figures 2.3-14 and 3.5-1, Figure 3.5-2 compares XMAX(8-puzzle, K_2 , W , N) values the corresponding XWORST(M , K_{MIN} , K_{MAX} , W , N) values for $W = .5$ and $W = .2$. Figure 3.5-3 is similar, using $W = .7$ and $W = 1.0$. Figures 3.5-4 and 3.5-5 are comparable to Figures 3.5-2 and 3.5-3, respectively, using heuristic K_1 instead of K_2 . Figures 3.5-6 and 3.5-7 are similar, for K_3 .

The extent of agreement between XMAX and XWORST values can be quantified by measuring, for each choice of K and W , the average difference between the values over the range of N . One possible measure is the root mean square (RMS), which sums the squares of the differences between the values. Since the values in this case span several orders of magnitude, however, the standard RMS value would be dominated by the XMAX and XWORST values for large N . As a measure of agreement that weights the differences more uniformly, we sum the squares of the differences between the logarithms of the XMAX and the XWORST values. Since $\log x - \log y = \log(x/y)$, the following formula measures the average factor of difference between XMAX and XWORST.

$$E(K, W) = 10 \uparrow \left(\left(\sum_{1 \leq N \leq N_{max}} \log^2 (XMAX(K, W, N) / XWORST(K, W, N)) \right) / N_{max} \right)^{-1/2}$$

where $N_{max}(K, W)$ is the maximum value of N for which XWORST(N) (and hence necessarily XMAX(N)) is known. So $E(K, W) = 1$ iff XMAX and XWORST are identical for each N . Similarly $E = 1.12$ indicates that over the range of N , XMAX and XWORST differ by about 12% on the average, and $E = 2$ indicates they differ by a factor of 2.

Figure 3.5-8 plots $E(K, W)$ for the data shown in Figures 3.5-2 through 3.5-7 and for other values of W not shown in those figures. Most of the values of $E(K_3, W)$ are so large as to not appear on the scale of Figure 3.5-8. For example, $E(K_3, .5) = 33.7$ and $E(K_3, 1.0) = 191.2$. The smallest observed values of E , 1.13 and 1.16, occur for $(K, W) = (1, .2)$ and $(1, .3)$, respectively. Note that $E(K_1, W) \leq E(K_2, W) \leq E(K_3, W)$ for each W for which data is available with two exceptions: $E(K_1, .1) > E(K_2, .1)$ and $E(K_1, .5) > E(K_2, .5)$. At present we can offer no strictly technical explanation of why this is the case. Figure 3.5-9 plots the data of Figure 3.5-8 and the remaining $E(K, W)$ data on an extended ordinate scale.

The data show that for the two never-overestimating 8-puzzle heuristics K_1 and K_2 , fairly good agreement is obtained for $0 \leq W \leq .5$ between predicted worst case number of nodes expanded and the observed maximum number of nodes expanded. The

agreement for K_1 and K_2 deteriorates for $W \geq .5$: the model predicts increasing cost with increasing W , whereas experimentally the maximum number of nodes expanded follows the mean number of nodes expanded in generally observing increasing cost with increasing W for some values of N , followed by decreasing cost with increasing W for larger values of N . Here apparently the model assumptions are too strong: extreme worst case behavior does not appear to occur in practice for large W (at least for these experiments). For K_3 , an overestimator, the agreement is very poor for most values of W , but improves for small W . Since reducing (increasing) W is equivalent to multiplying the $K(s)$ values by a scalar less than (greater than) one, the evidence indicates that DEBET predictions are fairly accurate for never-overestimating heuristics in this generalized sense, and relatively poor otherwise. This experimental test of the theory has thus served the useful purpose of indicating some of its current weaknesses.

3.5.2 Comments

The following factors contribute to the disagreement between predicted and observed values:

- 1) The 8-puzzle graph is not a uniform tree
- 2) None of K_1 , K_2 , or K_3 is identical to its corresponding KWORST function
- 3) the XMAX values are biased estimates
- 4) the KMIN(i) and KMAX(i) values are biased estimates

This enumeration raises the task of determining the extent to which each of the above factors to the aggregate disagreement. We leave this task to future work, but suggest here how this work might proceed. The techniques of order statistics (e.g., [Barlow, et al. 1972], [David 1970], [de Haan 1976], [Gumbel 1958]) are useful in determining how accurately observed maximum and minimum values obtained by sampling estimate the true max and min values in the distribution from which the sample is taken. Such techniques are likely the best available analytic means for determining the accuracy of the XMAX and KMIN and KMAX estimates. It would seem prudent to supplement such an analysis with experiments to measure directly the dependence of the XMAX, KMIN, and KMAX values on the number of samples.

The extent to which XWORST varies with KMIN or KMAX values is a property of the model itself, i.e., of the formula relating XWORST to KMIN and KMAX. Theorem 3.1-2 shows XWORST to be an exponential function of YMAX, hence a detailed sensitivity analysis of the model might indicate that XWORST is indeed ill-conditioned (defined in this case to mean that small changes in KMIN(i) or KMAX(i) values can cause relatively large changes in XWORST value).

It would seem at present more problematic to isolate and measure the contributions to the observed discrepancies of the differences between the DEBET model assumptions and the problem graph assumptions (i.e., factors 1 and 2). A first attempt at explaining the discrepancies might usefully concentrate on factors 3 and 4, ignoring factors 1 and 2.

Even if the predictions and observations were known always to be in close agreement, it is doubtful the current results could have much practical impact because: 1) practitioners may be more interested in average case performance than in worst case performance; 2) it may not be cost-effective to make predictions using a model that requires costly experiments to determine the values used as inputs to the model (i.e., the KMIN and KMAX values). Regarding the latter point we offer now an example indicating the possibility of deriving the KMIN and KMAX values analytically rather than by experiment. Let the PxQ-Knights-Move graph be an undirected graph having $P \times Q$ nodes, such that an edge connects two nodes iff the nodes, considered to be squares on a $P \times Q$ chessboard, are separated by a knight's move. To get from one given square to another, an obvious and very effective strategy is simply to move in the direction of the goal square until a few squares away, followed by one of a small number of specialized short move sequences. If the nodes of the knights-move graph are encoded in the obvious way as ordered pairs $s = (u,v)$ corresponding to rows and columns of the board, then an obvious K function for this graph is the rectilinear distance metric, namely

$$K_{\text{rect}}(s_1, s_2) = |u_1 - u_2| + |v_1 - v_2|$$

Figure 3.5-10 depicts a portion of a knights-move graph: each square in the figure represents a node of the graph and the number inscribed in each square indicates the minimum distance in the graph from that node to the node represented by the square inscribed with "0". Assuming for simplicity an indefinitely large board, inspection and a trivial analysis indicates that for this graph and K function we have $KMAX(i) = 3i$ for $i \geq 0$, $KMIN(i) = 3i - 8$ for $i \geq 3$ and $KMIN(1) = 3$ and $KMIN(2) = 2$, as

depicted in figure 3.5-11. (For $i \geq 3$, squares achieving $KMIN(i)$ appear in the same row or column as the goal square; squares achieving $KMAX(i)$ are near one of the diagonals that pass through the goal square.)

Assuming the 8-puzzle graph and heuristic K_1 instead of the knights-move graph and K_{rect} , it is similarly evident that in this case $KMAX(0) = KMIN(0) = 0$ and $KMAX(1) = KMIN(1) = 1$, but it appears difficult at present to extend such an analysis to derive these values for larger i . Also of possible relevance is Berliner's [1978] notion of state classes.

We note incidentally that K_{rect} for the knights-move graph is an uncontrived example for which the difference $KMAX(i) - KMIN(i)$ does not grow with i .

Summarizing this section, we have now complete the exercise of defining precise performance measures, measuring their values experimentally, deriving formulas to predict these values, and comparing the analytic predictions with the experimental observations. The results indicate that there is room for improving the accuracy of the predictions. In the case of $K = K_3$ and $W = 1.0$, an alternative model will better the DEBET model if its estimates of $XMAX$ are within a factor of 191 (by the $E(K,W)$ measure) of the actual values. For the case $K = K_1$ and $W = .2$, an alternative model's estimates must be no more than 12 percent off to better the DEBET prediction. Nevertheless, the DEBET model stands as the most accurate predictor to date of the experimental values of the sort given in Chapter 2. It is clear that more experimental data like those reported in Chapter 2 are needed to test further the practical applicability of the DEBET model.

3.6 Conclusions and Open Problems

The results reported in this chapter demonstrate the benefits and the limitations of a methodology that combines controlled experiment with rigorous theory. Our objective has been to narrow the gulf between the rigorous theory and the everyday practice of best-first search, but the results indicate that a considerable gap yet remains. This section highlights the technical results, and poses open problems for future work.

We have extended previous analytic results concerning the worst case cost of A^* search of uniform trees from a particularly simple special case, a class of heuristics characterized by two real scalar values, to a more general case, in which an arbitrary heuristic is modeled by two real-valued functions on the non-negative integers. The analysis has uncovered that the dominant factor in determining cost is the relative error in heuristic estimates of distance to the goal: for a broad class of never-overestimating heuristics, cost is a very simple exponential function of relative error. In particular, if relative error remains constant with increasing distance, then cost grows exponentially. Similar results were obtained identifying what relative error can be tolerated and still guarantee that cost grows no faster than sub-exponentially, polynomially, and linearly, respectively. Left somewhat unsettled, however, is the question of why worst case cost (e.g., for the general class of IM heuristic functions or the class of IM-never-overestimating heuristic functions) should be expressed more simply in terms of relative error than in terms of absolute error: what makes relative error more special? We know of no strictly mathematical reasons implying that a formula for XWORST expressed in terms of relative error functions need be more concise than one expressed in terms of absolute error.

Other results showed how performance varies with relative weight given to the heuristic term in the evaluation function: equal weighting is optimal for "IM-never-overestimating" heuristic functions. It was also shown, in the case of linearly-bounded heuristics, that this is not necessarily true for K functions that are not "IM-never-overestimating". More general versions of the optimal weighting question remain open at present, e.g., for arbitrary IM or DM heuristic functions, which value of W is optimal?

Experiment is the judge of the predictive ability of a theory. Here, the results are mixed. The comparison of predicted vs. measured data reported in Section 3.5 indicate that the DEBET model, despite its simplicity, predicts XMAX values accurately within a factor of 10 across most of the test range of heuristic functions, values of W , and values of N . (The best prediction registered a 12% error.) However, its predictions are not uniformly accurate enough to be applied in practice, e.g., to decide which of K_2 and K_3 for the 8-puzzle is more efficient or which value of W minimizes cost. On the other hand, the disagreement itself reveals a new fact about the phenomena of best-first search: extreme worst case behavior does not appear to occur in practice for large W or for certain heuristics. It may be interesting to refine the DEBET model so that the $K(s)$ values for nodes on the solution path of the uniform

tree are not restricted to be the $KMAX(i)$ values of K , but rather may match more closely those that occur in practice. In any case, since the DEBET model applies to any $KMIN$ and $KMAX$ functions, it will be interesting to test its predictions for problems and heuristics other than the 8-puzzle case study used here. Of course, to perform such a test requires experimental $KMIN$, $KMAX$, and $XMAX$ data of the sort reported in Chapter 2.

It would be interesting to obtain comparable results for average case cost, especially to determine whether optimal weighting occurs at $W = 1$, as in the experimental results, as opposed to $W = .5$ for these worst case results. Preliminary investigations indicate that simple general formulas are even more difficult to obtain in closed form for average case than for worst case. However it seems possible to incorporate non-closed form analytic results into a program, as in Section 3.5, that calculates numerically the average number of nodes expanded, given as input the $KMIN(i)$ and $KMAX(i)$ values. Although less satisfying than a closed form result, such a numerical evaluation approach has at least the advantage that quantitative answers can be obtained for any particular case at an insignificant fraction of the cost required to actually execute a set of searches as in Chapter 2. Hence this approach may prove useful for discovering and testing hypotheses that may be theorems.

Because here a problem is a uniform tree, the current model cannot account for the observation that some problem graphs seem intuitively to be more complex than others. For the class of problem graphs such as the 8-puzzle, the question can be phrased in terms of a hypothetical relation between heuristic search performance and problem "structure".

Within the DEBET model as presently defined, it would be interesting to expand the set of monotonicity theorems (i.e., Theorems 3.2-5, 3.2-6, 3.2-7, 3.3-3, and 3.3-4) to include additional pairs of $\langle KMIN, KMAX \rangle$ not already covered by one of those theorems. For example, we conjecture that if W is given such that $K_1 = \langle KMIN_1, KMAX_1 \rangle$ is IM at W and we let $K_2 = \langle KMIN_2, KMAX_2 \rangle$ such that $KMIN_2(i) = i \cdot KMIN_1(i) / KMAX_1(i)$ and $KMAX_2(i) = i$, then for all M and N , $XWORST(M, K_2, W, N) \leq XWORST(M, K_1, W, N)$. (This is a generalization of the scalar optimization operation defined in Section 3.2 for $\langle a, b \rangle$ heuristic functions.)

Variations of the DEBET model can be defined simply by changing the range of the K functions, e.g., from $\mathbb{R}^+$ to $\mathbb{N}$. Given this restriction, are any stronger statements

provable? Similarly, the model suggests types of heuristics unlike those investigated in the past, e.g., by changing the dimensionality of K functions to take as arguments three nodes of $T(M, N)$ (the third being the goal node u_N), and return a binary value indicating with certainty whether the first node is closer to the goal than the second node. It may be problematic to devise such a heuristic in practice, however it may be easier to do so if a tri-valued range is permitted, e.g., $(0, 1, ?)$ where "?" denotes no information for these arguments. Alternatively, $(0, ?)$ and $(1, ?)$ are possible ranges for a K function, as are ordered sets having a small number of elements. (An additional restriction might be that the nodes s_i and s_j representing the first two arguments be such that $h(s_i, s_j) \leq 2$, e.g. as would be the case if s_i and s_j are successors of the previously expanded node.) This sort of heuristic function would give exact information in a subset of cases rather than inexact information (in general) for all cases.

3.7 DEBET Results as a Step Toward a Theory About the Relation of "Heuristic Knowledge" to Performance

In this addendum to Chapter 3 we proffer several informal interpretations of the technical results presented in the preceding sections of this chapter, viewing those results as an instance of a formal (but limited) theory within a restricted technical context about "knowledge" and its relation to problem solving performance. Our approach is to ask: If there were to exist a mathematical theory about "knowledge", what might it tell us? In what sort of terms might its statements be expressed? In answer, we highlight certain characteristics of the DEBET model that may serve as possible guidelines in developing such a theory. Although informal, these comments are intended to convey in as precise a form as is now possible a set of constraints within which work toward a rigorous theory of "knowledge" might proceed.

The experience of AI researchers with knowledge-based systems can be summarized by the statement "Expert knowledge buys expert performance" (see e.g., [Feigenbaum 1977]). Determining exactly how much knowledge buys how much performance is problematic, however, because a rigorous theory would require that "knowledge" and "performance" be well-defined, measurable quantities. Whereas attempts have been made to measure the performance of certain AI systems under varying conditions (e.g., [Paxton 1976] and others), it is not yet clear how to measure

or even delimit, in a uniform and objective way, the "knowledge" possessed by such programs. Our approach in this chapter has been to obtain precise answers in a technical context that is impoverished relative to the sophistication of contemporary AI systems.

First we observe that DEBET says nothing about "knowledge" per se, but only about the operational consequences of "knowledge". That is, the "knowledge" embodied in a heuristic function K is such that K 's distance estimates are bounded by the values given by its characteristic $KMIN$ and $KMAX$ functions. We posit a formal model of "knowledge" itself in which there are at least as many distinct "states of knowledge" as there are distinct $\langle KMIN, KMAX \rangle$. That is, if the possible "states of knowledge" can be delimited as a set, then the cardinality of that set is not less than the cardinality of KB^* . Henceforth we speak only of KB^* .

DEBET specifies the relation between "knowledge" and performance by defining on the set KB^* a function (i.e., $XWORST$) whose range is another set consisting of functions of the form $\mathbb{N} \rightarrow \mathbb{N}$ (i.e., the set, call it P of performance functions mapping the size of the problem to the number of steps executed). (For simplicity, we ignore W .)

This distinction between the knowledge set and the performance function set suggests that theories about "knowledge" of the sort discussed here distinguish the "knowledge" from the "knowledge engine". In DEBET the A^* "knowledge engine" is a parameterized problem solving mechanism whose performance in solving a class of problems varies with (i.e., is a function of) the "knowledge" it is given. The set KB^* exists independently of A^* ; it happens to be the domain of a particular function we have called $XWORST$. We say that a "knowledge engine" is comparable to A^* if it induces a function having the same domain and range as $XWORST$, and there as many such mutually comparable "knowledge engines" as there are distinct functions from KB^* to P . For example, ordered depth-first search and the B^* algorithm schema [Berliner 1978] are comparable to A^* in this sense, since the heuristics each can use can be modelled by $\langle KMIN, KMAX \rangle$ functions, and each causes nodes to be expanded. (As defined in Chapter 2, ordered depth-first search is identical to A^* under the restriction that the next node expanded must always be selected from among the successors of the last node expanded.) Hence an interesting open problem is to derive a similar $XWORST$ function, assuming as the "knowledge engine" ordered depth-first search or

the B^* algorithm instead of A^* . If you give an engine more knowledge, i.e., less errorful knowledge, then it performs better (i.e., the monotonicity results). But some engines can do more than others (or do it faster) with the knowledge they are given. Hence in our distinction, "knowledge" is equated with the set KB^* and a "knowledge engine" is equated with a particular mapping from KB^* to P .

This distinction between "knowledge" and "knowledge engine" suggests a possible analog to the concept of IQ. It would seem natural that the study of artificial intelligence should include the concept of the "IQ" of an intelligent mechanism, but quantifying the concept is problematic in the general case. (Lest the term "IQ" connote inappropriate human-like qualities, we suggest the term "Performance Capability" or "PC" in the context of machine intelligence.) The IQ value for a human is defined by a scalar quantity measuring his or her performance for one particular state of knowledge, namely that possessed at the time of the test. (We consider IQ here more as an absolute measure than as a relative measure based on age comparisons.) The human's IQ does not indicate what his performance would be given more or less or different knowledge than that possessed at the time of the test. Hence person A may score higher than person B simply because A has a larger or more accessible store of relevant knowledge than does B, rather than because A is inherently more capable than B, but a single IQ test cannot distinguish these two cases. In contrast, we can define the PC of a problem solving mechanism as a quantitative measure of its performance as a function of the "knowledge" it is given, e.g., the XWORST mapping from knowledge set to performance set. The XWORST mapping is like a function $z = f(x, y)$ that describes a surface in 3-space above the x-y plane. Each of the axes (one corresponding to KMIN, one to KMAX, and the vertical axis to XWORST), however, is not a linear ordering of integers or reals, but rather an infinite continuous lattice of functions on the non-negative integers. Hence the PC (i.e., in this case, XWORST) maps each point in the "plane" of "knowledge" to its corresponding value of performance. As different human individuals have different IQs, so with the PCs of different problem solving mechanisms, if they are comparable in the sense defined in the preceding paragraph. The general usefulness of a quantitative definition of the PC of a "knowledge-parameterized" problem-solving mechanism remains to be demonstrated. As used here, it is simply an interpretation for particular mathematical results.

The DEBET results furthermore reveal some of the difficulties inherent in attempting to formalize complex quantities such as "knowledge". Here the "knowledge"

of a heuristic is modeled by the functions that bound the values it computes, and the difficulties arise in attempting to manipulate a function of functions. For example, the results that cost increases monotonically with relative error (i.e., Theorems 3.3-3 and 3.3-4) suggest it would be interesting to determine the rate at which cost increases per unit increase in error. (In familiar terms, if the heuristic is improved a little, does cost decrease a little or a lot?) If these were continuous scalar quantities, we would simply take the derivative of cost with respect to error. However, the fact that both cost and error are functional quantities makes problematic even a precise formulation of the question. Our efforts at defining partial orderings on KB^* may be relevant toward progress of this sort.

Finally we note that the linear/polynomial/exponential results reported in Section 3.3 would seem to be interesting in light of two recent instances, in speech understanding [Medress 1977] and chess playing [SIGART 77], of superior performance attained by relatively simple mechanisms, HARPY and CHESS 4.5 respectively, that rely more on extensive and efficient search than on "knowledge" about the problem domain. (See also [Siklossy et. al. 1973], which compares a breadth-first search program with the Logic Theorist.) These instances, while possibly coincidental, support a hypothesis that "simple suffices", i.e., that for many tasks there exist simple mechanisms giving excellent performance, implying that in these cases "expert knowledge" is not a necessity. The results given in this chapter can be interpreted as one bit of evidence to the contrary, since guaranteed good performance of A^* (a simple mechanism) requires an extremely accurate heuristic. Since these are worst case rather than average case results, great weight cannot be granted this evidence.

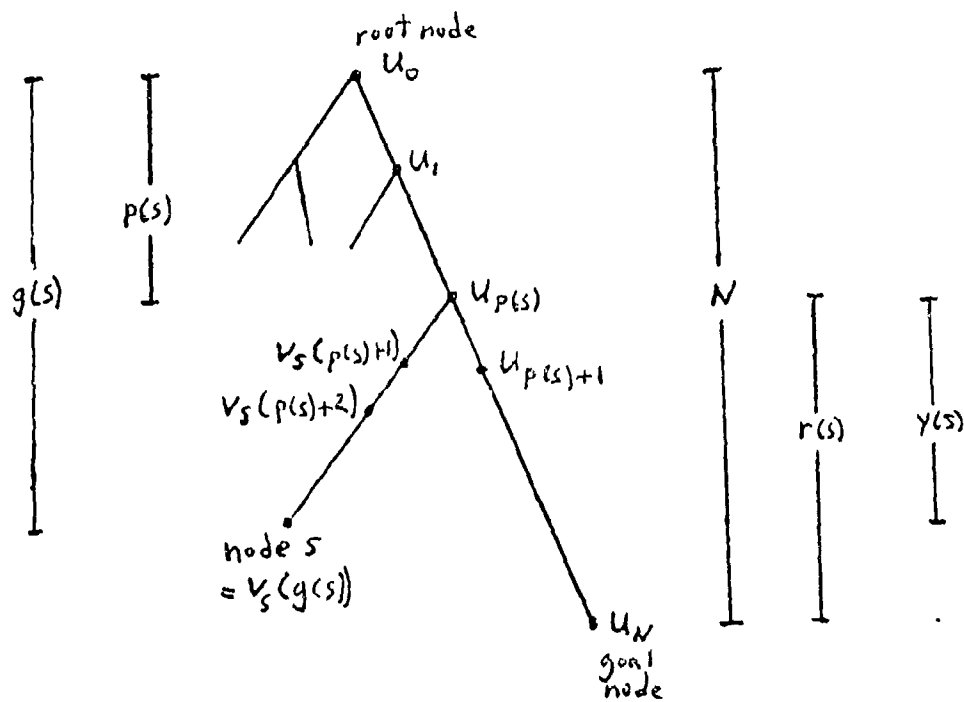


Figure 3.1-1 Tree notation

N is the depth of the goal node:

the nodes on the solution path are $u_0, u_1, \dots, u_{N-1}, u_N$

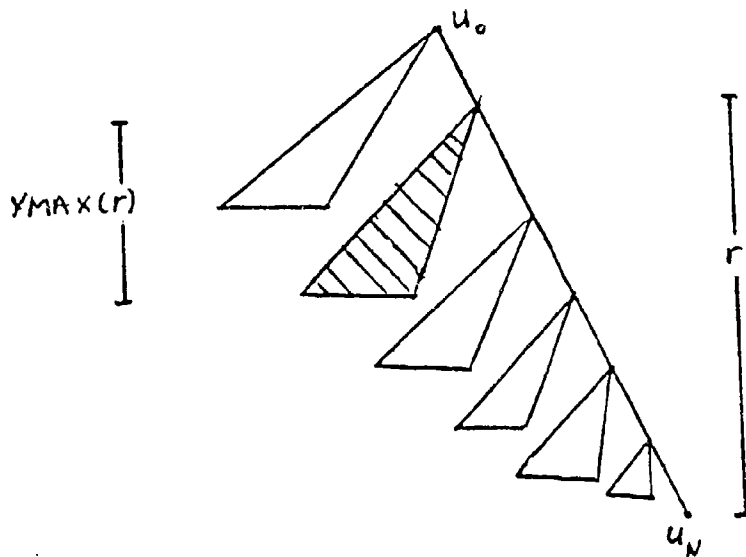


Figure 3.1-2 In the DEBET model each heuristic function causes certain subtrees of nodes to be expanded in the worst case. Each such subtree of expanded nodes is a uniform tree rooted at one of the nodes on the solution path. The depth of the subtree rooted at the solution path node that is r steps from the goal node is given by $YMAX(r)$. Later we define $YMAX$ as a function of the heuristic bounding functions $KMIN$ and $KMAX$ and of a scalar weighting value W , as well as of r .

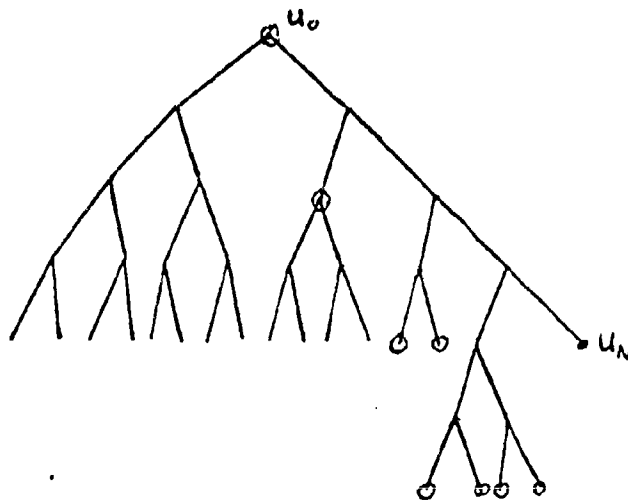


Figure 3.1-3 Nodes at distance 4 from the goal node, for a uniform tree $T(2,4)$ having branching factor $M = 2$ and depth of goal $N = 4$.

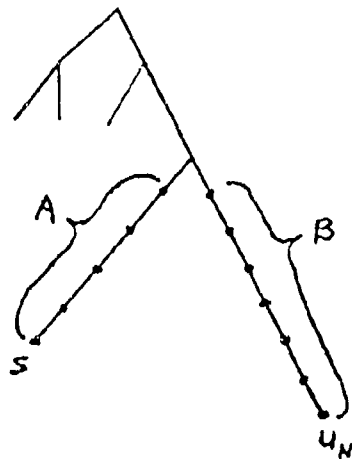


Figure 3.1-4 For an arbitrary evaluation function $F(s)$ such that A^* terminates using that function, an arbitrary node s not on the solution path is expanded iff the maximum F value for nodes in set A is less than or equal to the maximum F value for nodes in set B .

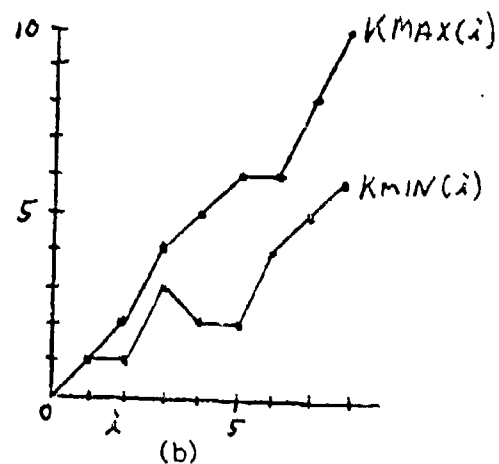
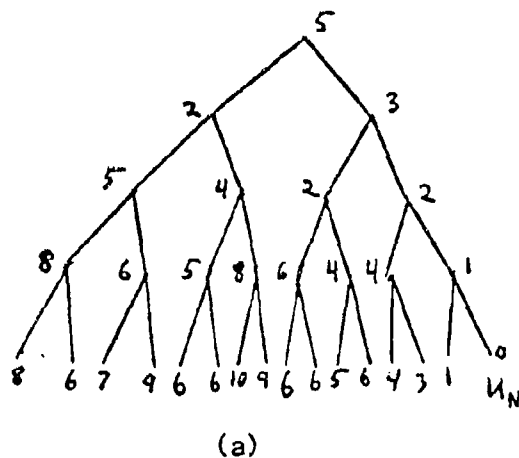


Figure 3.1-5 (a) Hypothetical $K(s)$ values for nodes in a portion of $T(2,4)$.
 (b) $KMIN(i)$ and $KMAX(i)$ values corresponding to these $K(s)$ values.

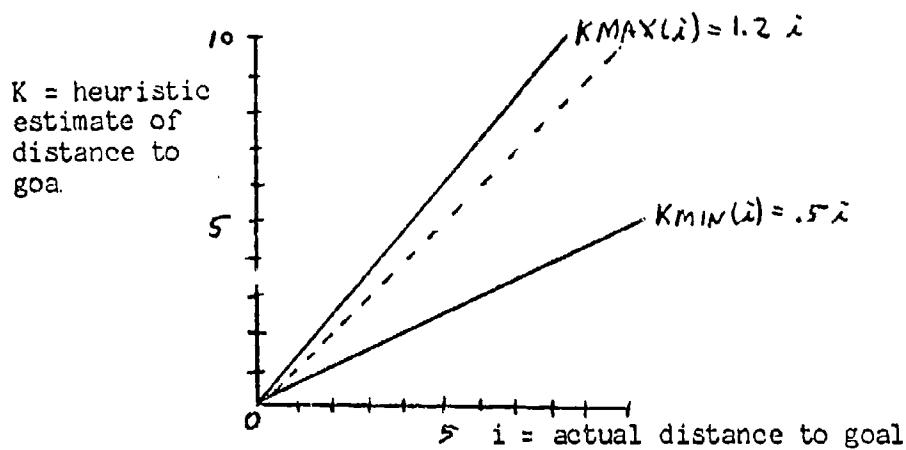


Figure 3.1-6 Bounds on heuristic estimates of distance to goal vs. actual distance to goal

An example of a "linearly-bounded" heuristic function, taking the form $K_{MIN}(i) = a i$ and $K_{MAX}(i) = b i$. In this case $a = .5$ and $b = 1.2$. Hence we identify this heuristic function as $\langle a, b \rangle = \langle .5, 1.2 \rangle$

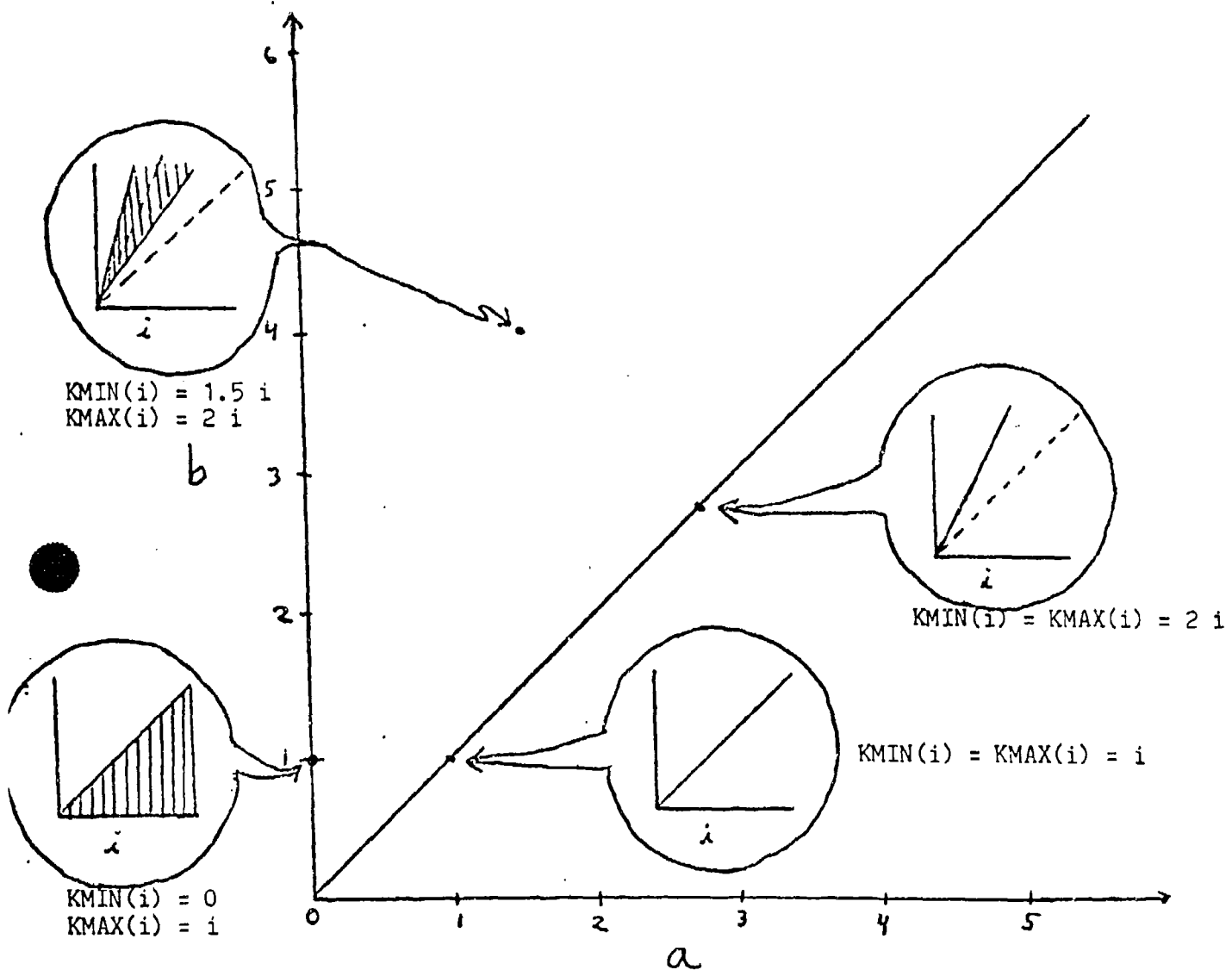


Figure 3.1-7 The set of all "linearly bounded" heuristic functions represented as the portion of the Euclidean plane for which $b > a$. The point (a, b) in the plane corresponds to the heuristic function $\langle a, b \rangle$ having $K_{\min}(i) = a i$ and $K_{\max}(i) = b i$. In the upper left "blowup", the average of the heuristic distance estimates are greater than the actual distances. The estimates in the blowup at upper right are the same as the average estimates at upper left, but with no variation.

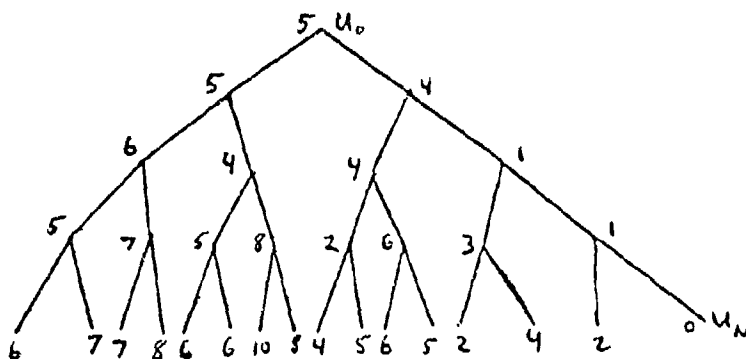


Figure 3.1-8 Analogous to Figure 3.1-5a. The $K(s)$ values here differ from those in Figure 3.1-5a, but correspond to the same $KMIN(i)$ and $KMAX(i)$ values as in Figure 3.1-5b.

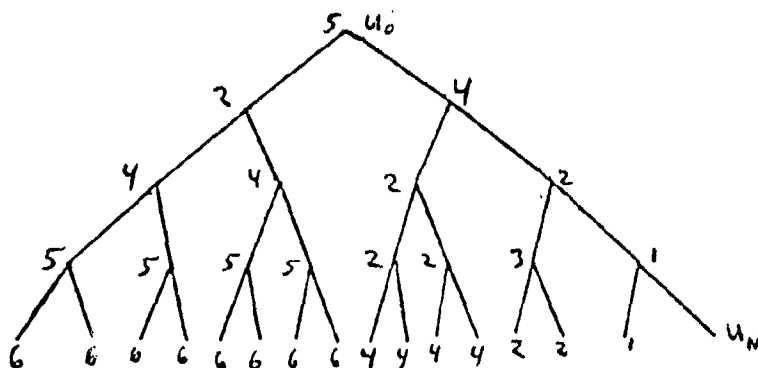


Figure 3.1-9 KWorst(s) values corresponding to the KMin(i) and KMax(i) values in Figure 3.1-5b.

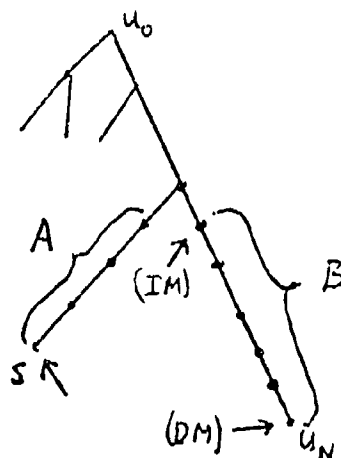


Figure 3.2-1 If a heuristic function $\langle KMIN, KMAX \rangle$ satisfies the "IM/DM" conditions and the evaluation function takes the form $F(s) + (1 - W) g(s) + W K(s)$, then the maximum F value for nodes in set A is $F(s)$, and the maximum F value for nodes in set B is $F(u_N)$ or $F(u_{p(s)+1})$, the former if $\langle KMIN, KMAX \rangle$ is IM, the latter if it is DM.

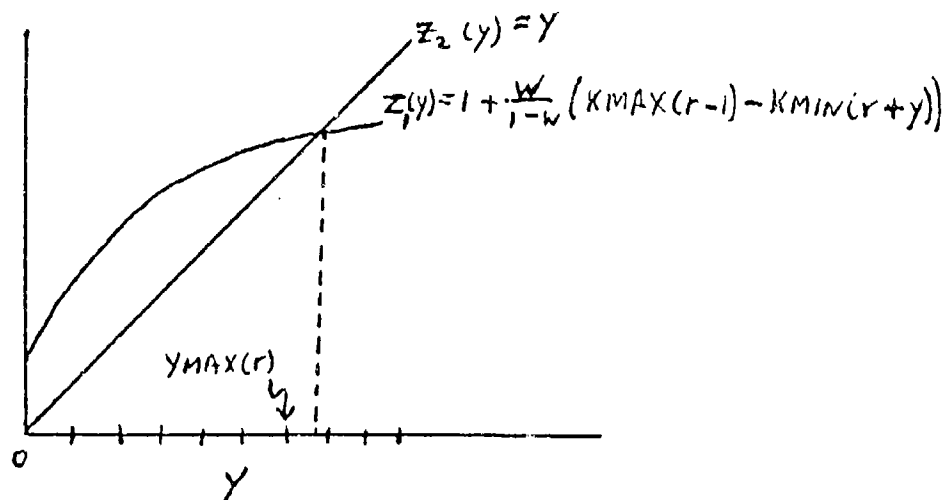


Figure 3.2-2 Geometric interpretation of computation of $YMAX(r)$. Given fixed values for $KMIN(i)$, $KMAX(i)$, W , and r , the value of $YMAX(KMIN, KMAX, W, r)$ is the absolute value of the value of y at which the real-valued function $z_1(y) = 1 + \frac{W}{1-W} (KMAX(r-1) - KMIN(r+y))$ first intersects the line $z_2(y) = y$.

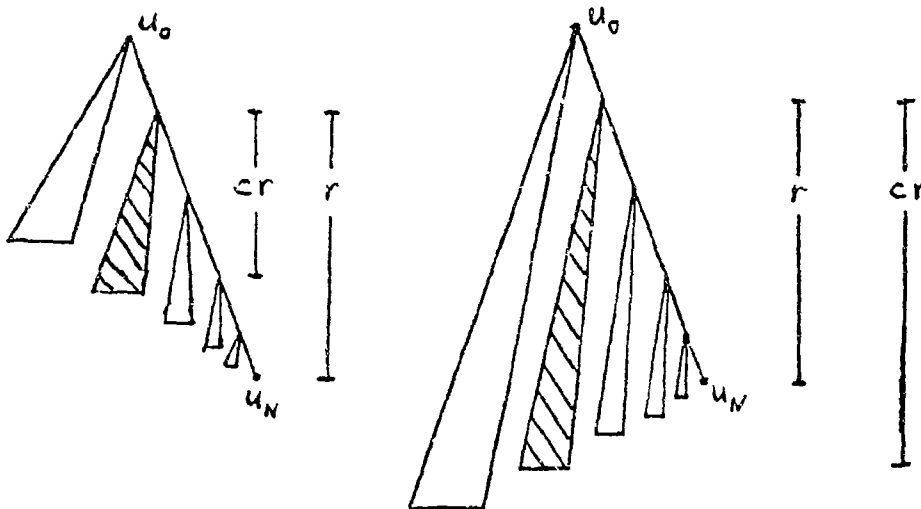


Figure 3.2-3 Subtrees of expanded nodes under the condition that the length of a "garden path" is proportional to the distance from the root node of the subtree to the goal node, i.e., $Y_{MAX}(r) = c r$. All linearly-bounded heuristic functions take this form.

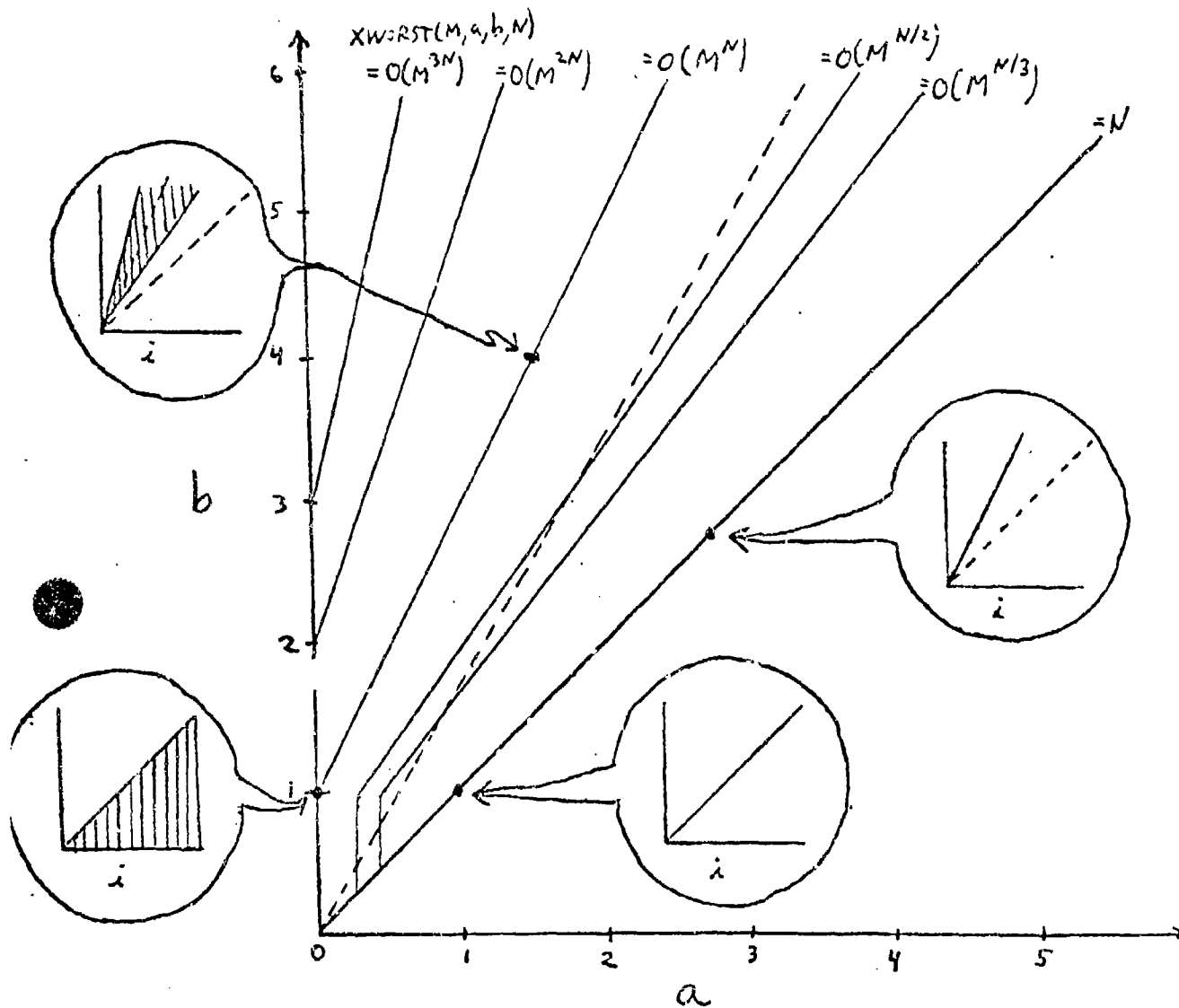


Figure 3.2-4 Iso-cost contour lines for linearly bounded heuristic functions
 Note that $XWORST(M,a,b,N) = N$ for functions $\langle a,b \rangle$ such that $b=a \geq 1$

If $b \geq 1$ then $XWORST(M,a,b,N) = O(M^{\lfloor N(b-a)/(1+a) \rfloor})$

If $b \leq 1$ then $XWORST(M,a,b,N) = O(M^{\lfloor N(1-a)/(1+a) \rfloor})$

A straight line through the origin (dashed line) corresponds to multiplying a and b by the same factor.

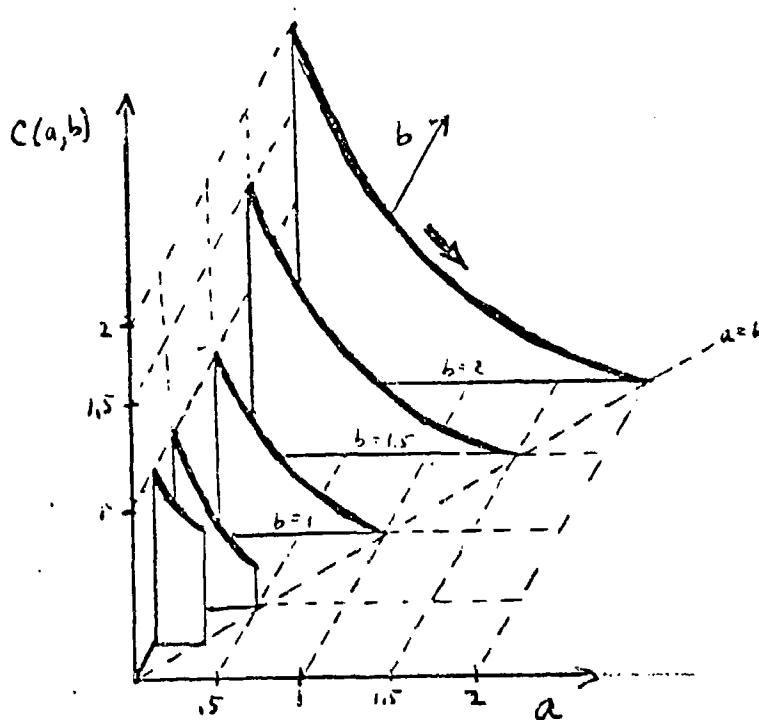


Figure 3.2-5

$$\text{Let } C(a,b) = \begin{cases} \frac{b-a}{b+a} & \text{if } b \geq 1 \\ \frac{1-a}{1+a} & \text{if } b \leq 1 \end{cases}$$

Then $XWORST(M,a,b,N)$

$$= O(M^{C(a,b)} N^{\downarrow})$$

The function $C(a,b)$ defines a surface above the a,b plane. The heavy lines plot contours of the $C(a,b)$ surface for the line segments $b = .25$, $b = .5$, $b = 1$, $b = 1.5$, and $b = 2$. A point on the $C(a,b)$ surface corresponds to an instantiation of $XWORST(M,a,b,N)$ for fixed a and b as a function of branching factor M and depth of goal N . Note that the diagram is drawn to scale.

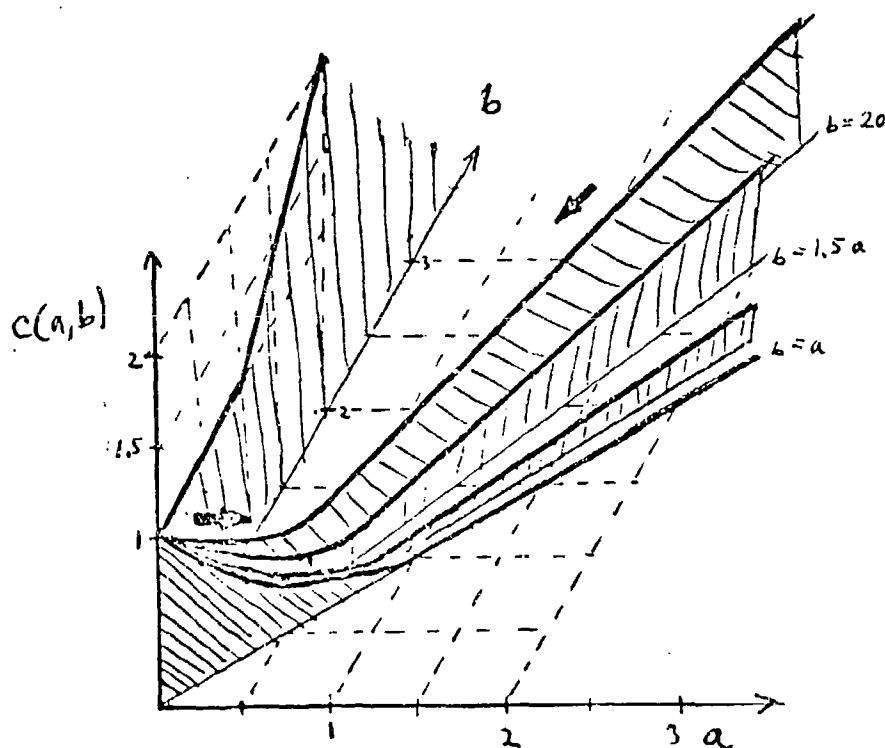


Figure 3.2-6

Plots of contours of the $C(a,b)$ surface for the rays $b=a$, $b=1.5a$, $b=2a$, and $a=0$. The arrows indicate direction of decreasing $C(a,b)$. Multiplying a and b by the same factor is equivalent to moving along one such contour. The diagram is drawn to scale.

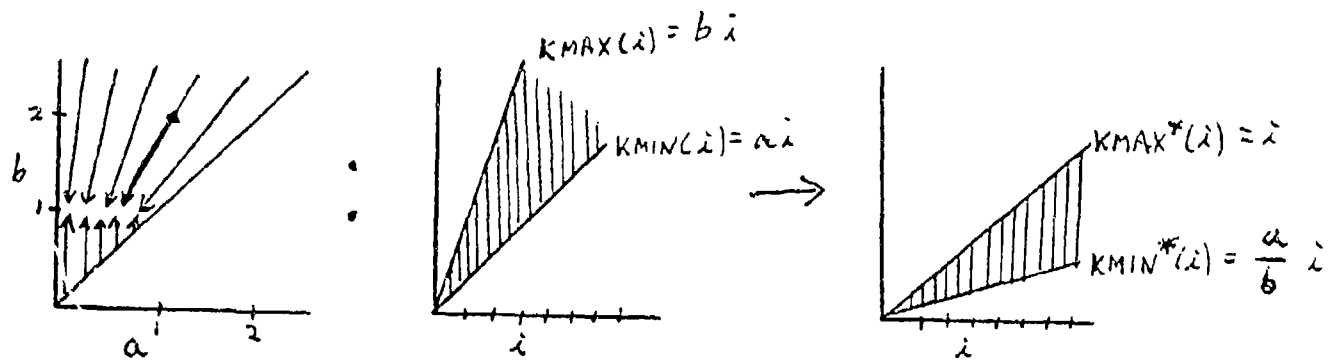


Figure 3.2-7 Linearly bounded heuristic functions $\langle a, b \rangle$ can be "scalar optimized" by dividing a and b by the value of b , resulting in a never-overestimating heuristic function. The diagram on the left depicts the scalar optimization operation as an autojection on the a, b plane. The middle diagram shows the particular function $\langle a, b \rangle = \langle a, 2 \rangle$, which scalar optimization maps to the function $\langle a, b \rangle = \langle .5, 1 \rangle$, depicted at right.

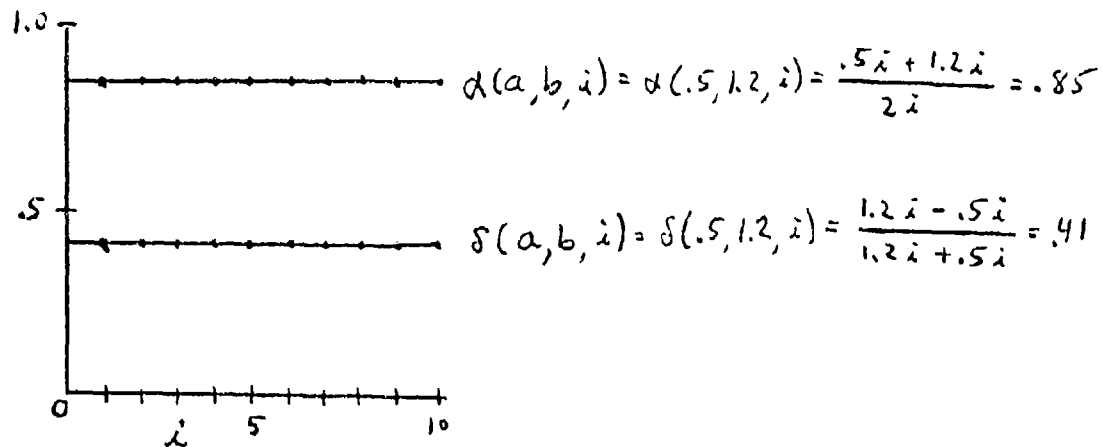


Figure 3.3-1 $\delta(i)$ and $\alpha(i)$ values for the linearly bounded function $\langle a, b \rangle = \langle .5, 1.2 \rangle$ shown in Figure 3.1-6. For linearly bounded functions $\delta(a, b, i)$ and $\alpha(a, b, i)$ are independent of i , i.e., the relative error and mean value factor are independent of distance to the goal.

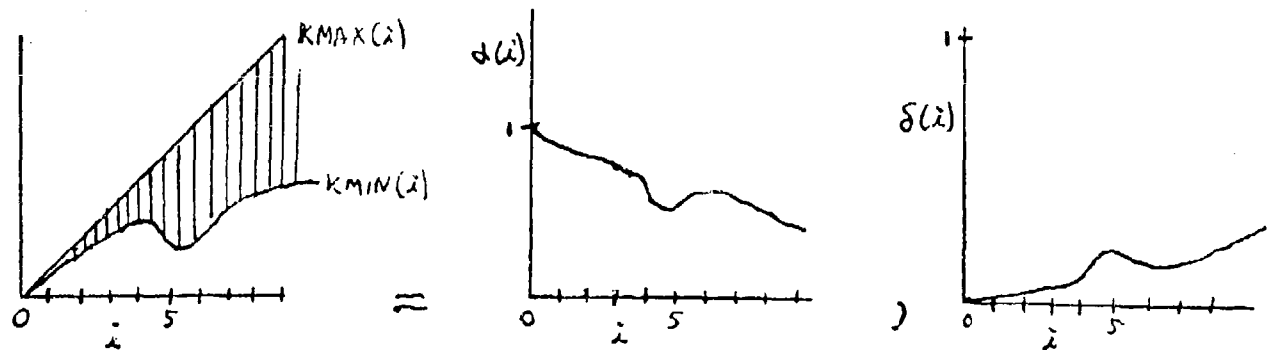


Figure 3.3-2 An example of the equivalence between the $\langle KMIN, KMAX \rangle$ and $\langle \alpha, \delta \rangle$ specifications of a heuristic function

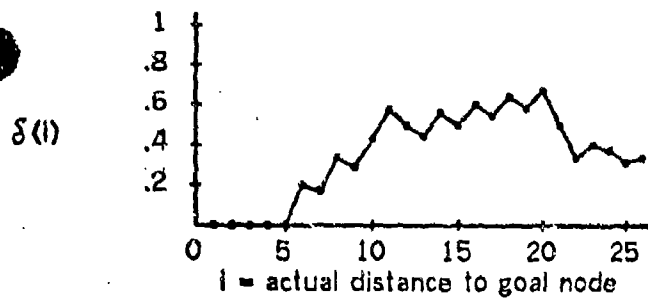
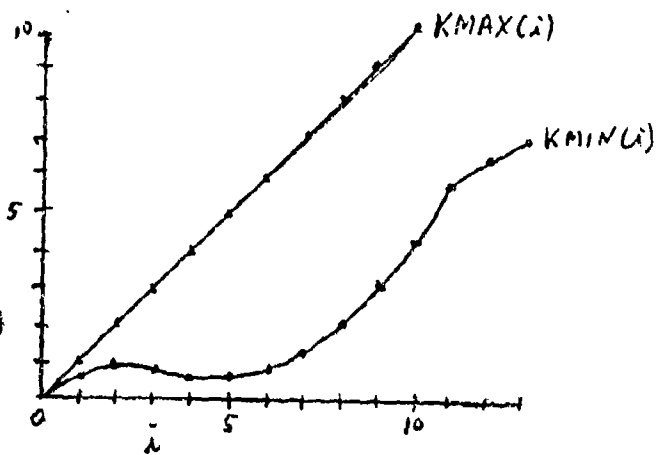
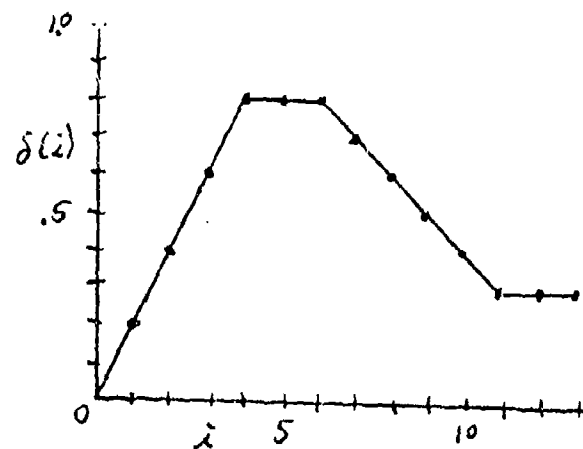


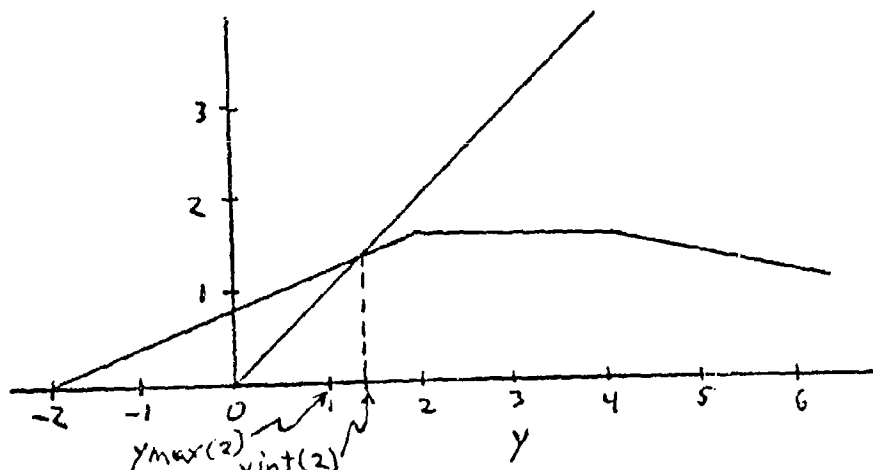
Figure 3.3-3 $\delta(i)$ = Relative error function for K_2
(Derived from $KMIN(i)$ and $KMAX(i)$ data in Figure 2.4-3)



(a) $\langle KMIN, KMAX \rangle$ form

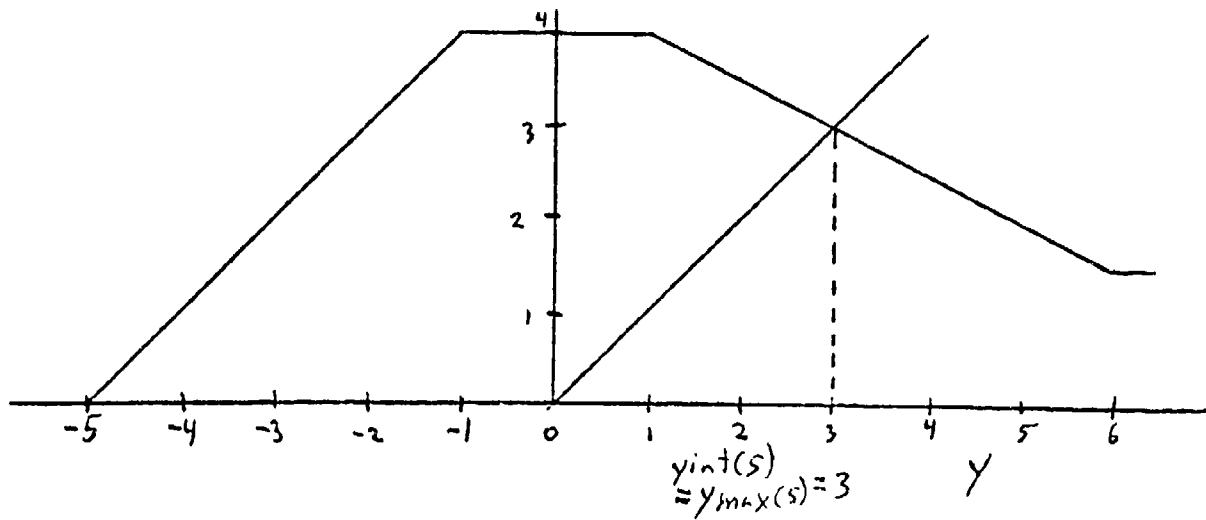


(b) equivalent $\delta(i)$ form

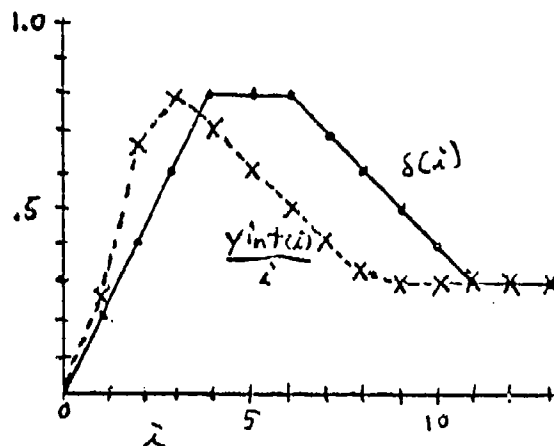


(c) Computation of the value of $YMAX(2)$ for the function depicted in (a) and (b). $YMAX(2) = \lfloor YINT(2) \rfloor = 1$

Figure 3.3-4 Geometric computation of $YMAX(i)$ from $\delta(i)$ for a particular hypothetical heuristic function that is "IM-never-overestimating"
(See text. See also Figure 3.2-2.)



(d) Geometric computation to determine that $YMAX(5) = yint(5) = 3$



(e) Relation between relative error $\delta(i)$ and $yint(i)/i$
 Note that $yint(2)/2$ and $yint(5)/5$ are determined from diagrams (c) and (d).

Figure 3.3-4 (continued)

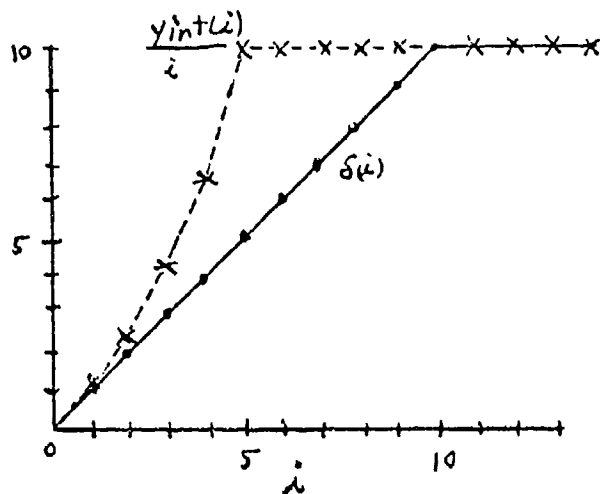


Figure 3.3-5 The relation between relative error $\delta(i)$ and $y_{int}(i)/i$ for a case in which $\delta(i)$ increases monotonically with i . (See Theorem 3.3-2.)

K = heuristic estimate of distance to goal

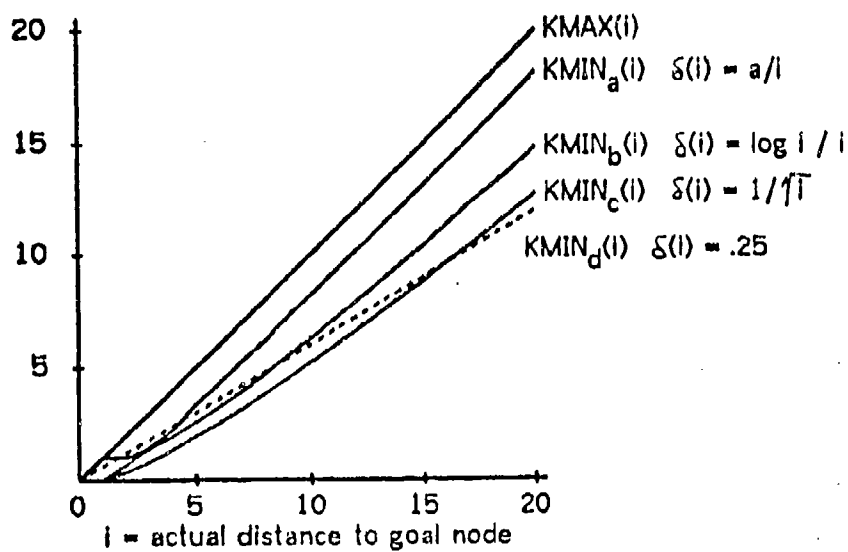


Figure 3.3-6 Comparison of the $KMIN(i)$ functions corresponding to different $\delta(i)$ functions (with $KMAX(i) = i$ in each case)

The $KMIN(i)$ functions have different asymptotic growth rates.

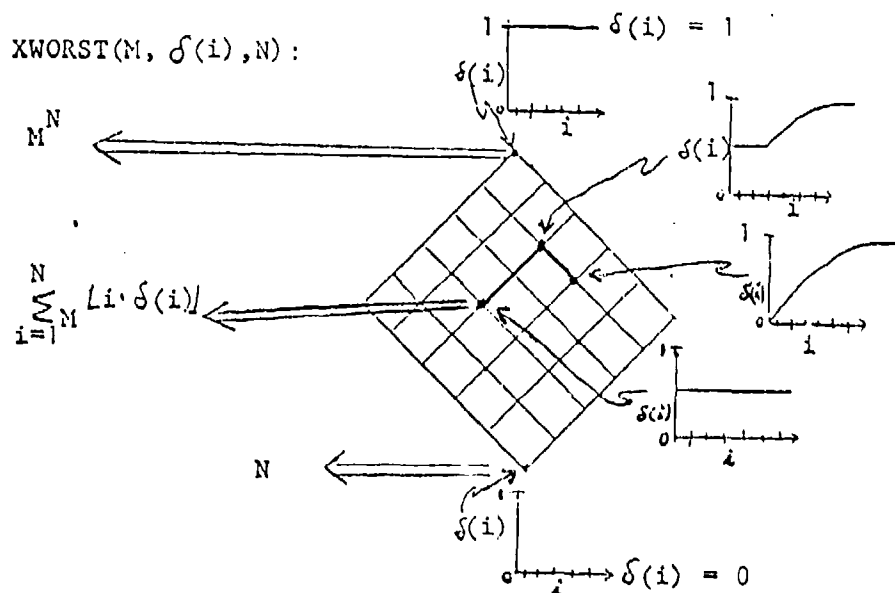


Figure 3.3-7 The class of IM-never-overestimating heuristic functions depicted schematically as a lattice. (Unlike this schematic lattice, the actual lattice is an infinite continuous lattice.) XWORST is defined as a particular function on this lattice

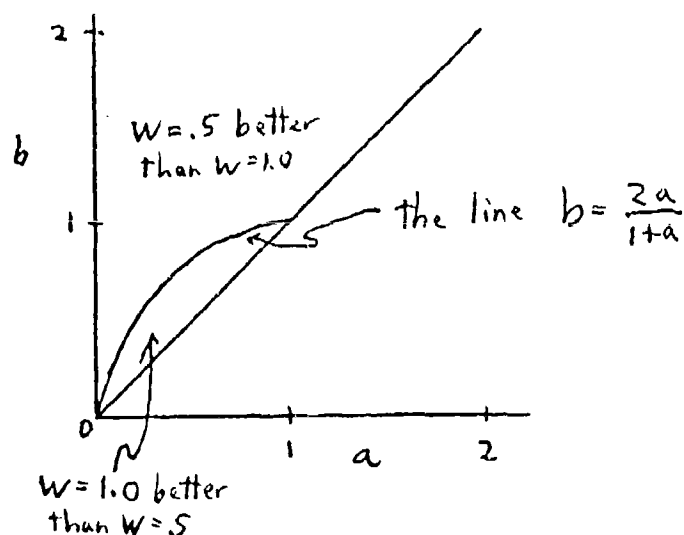


Figure 3.4-1 For the class of linearly-bounded heuristic functions, the line $b = 2a/(1+a)$ divides those functions for which $W = .5$ gives smaller XWORST cost (for all N) than $W = 1.0$ from those functions for which $W = .5$ gives larger cost than $W = 1.0$. (See also Figures 3.2-4 and 3.2-6.)

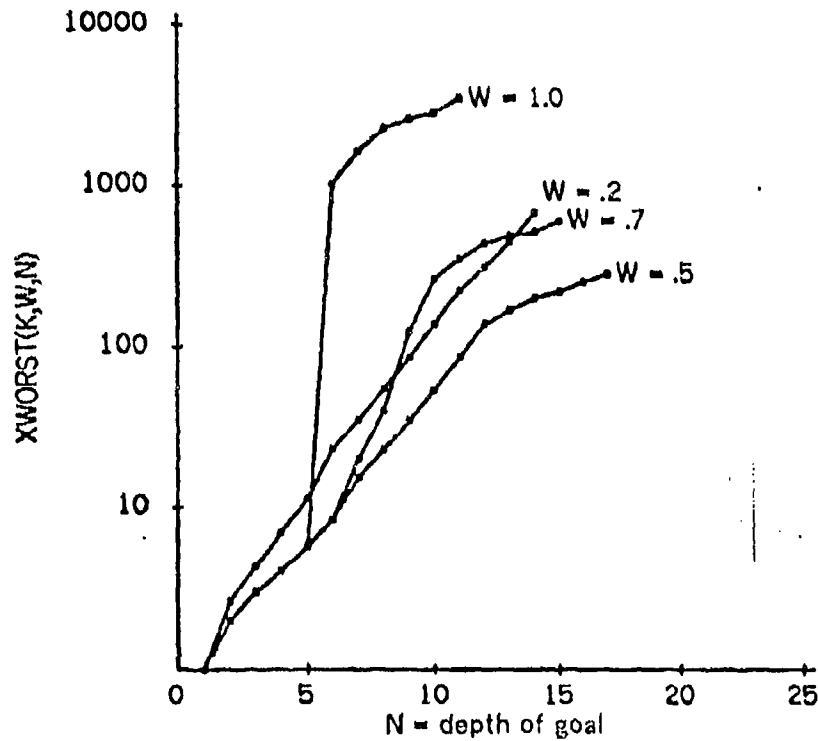


Figure 3.5-1 Predicted worst case number of nodes expanded for heuristic K_2 based on $KMIN(I)$ and $KMAX(I)$ data from Figure 2.4-3

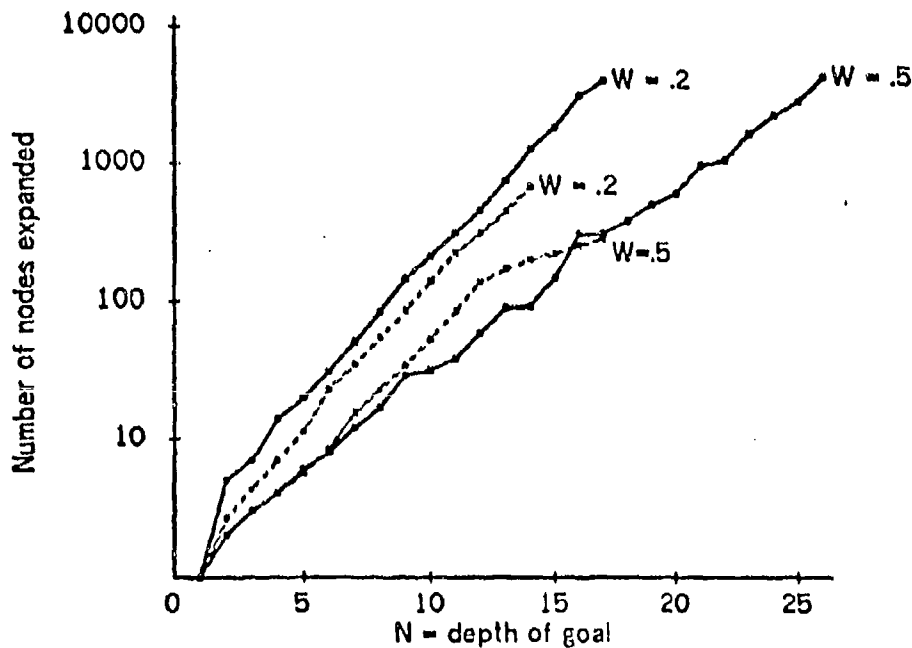


Figure 3.5-2 Predicted vs. observed number of nodes expanded in worst case for 8-puzzle heuristic K_2
 $XWORST(K_2, W, N)$ is predicted (dash), $XMAX(K_2, W, N)$ is experimental (solid)
 Each data point on solid curves based on up to 40 algorithm executions (a.e.)
 895 a.e. total for $W = .5$ (solid curve), and 640 a.e. total for $W = .2$ (solid curve)

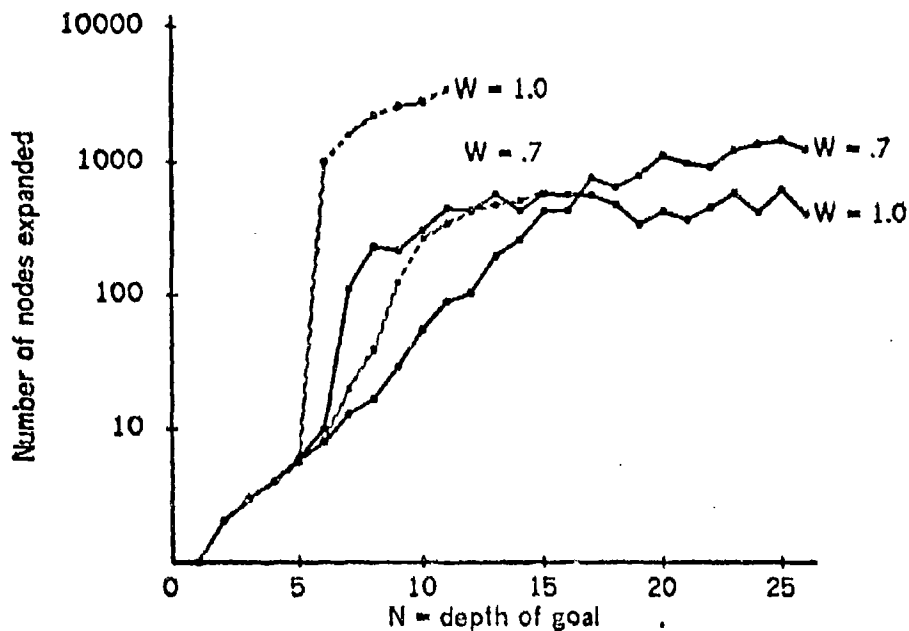


Figure 3.5-3 $XWORST(K_2, W, N)$ and $XMAX(K_2, W, N)$ for K_2 , different values of W . $XWORST(K_2, W, N)$ is predicted (dash), $XMAX(K_2, W, N)$ is experimental (solid). Each data point on solid curves based on up to 40 algorithm executions (a.e.) 897 a.e. total for each of $W = .7$ (solid curve), and $W = 1.0$ (solid curve).

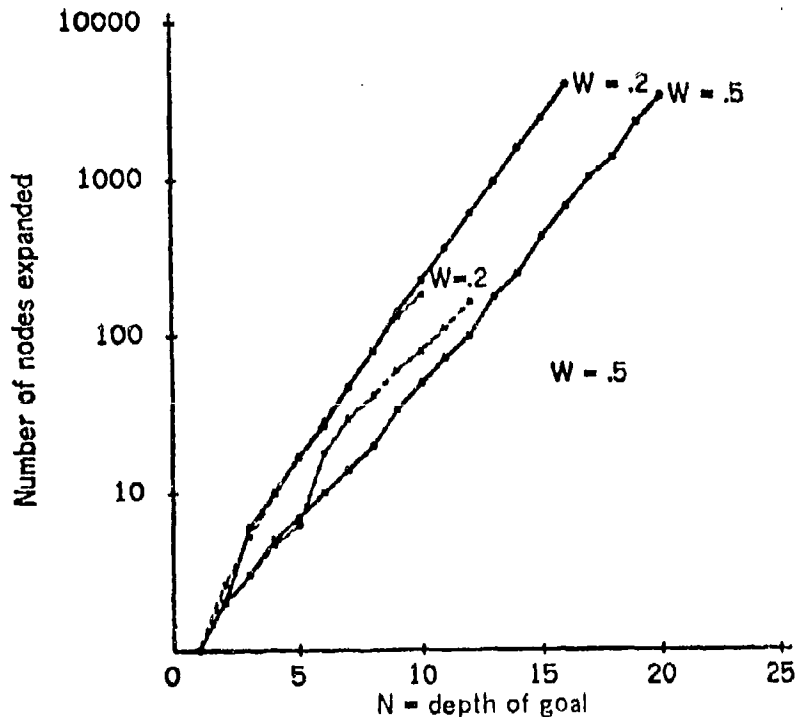


Figure 3.5-4 Analogous to Figure 3.5-2, for heuristic K_1 . $XWORST(K_1, W, N)$ is predicted (dash), $XMAX(K_1, W, N)$ is experimental (solid). Each data point on solid curves based on up to 40 algorithm executions (a.e.)

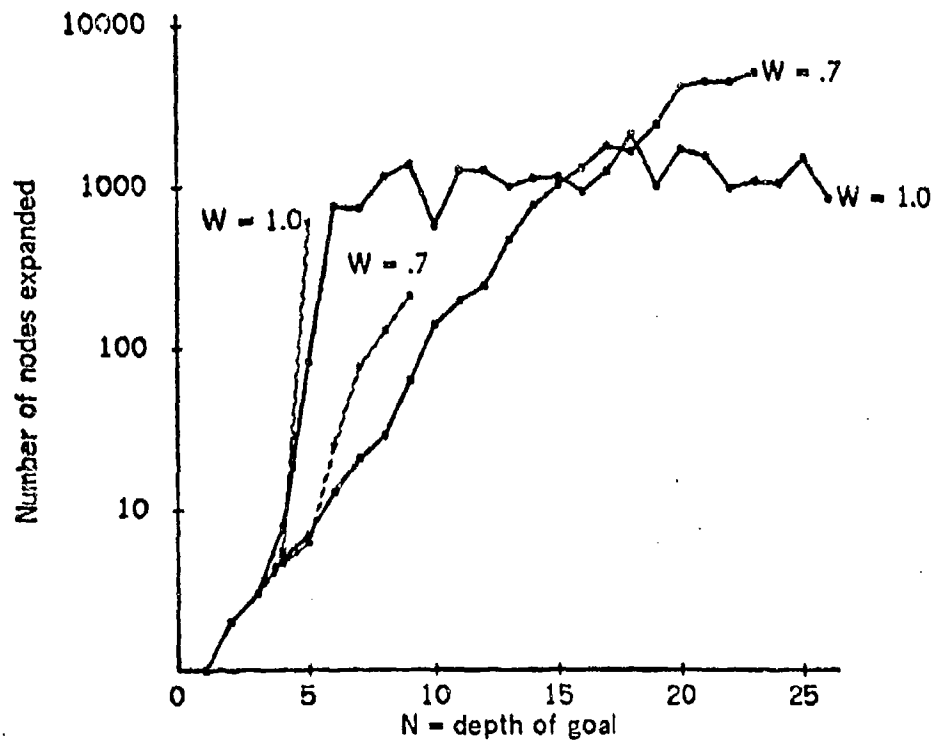


Figure 3.5-5 Analogous to Figure 3.5-3, for heuristic K_1
 $XWORST(K_1, W, N)$ is predicted (dash), $XMAX(K_1, W, N)$ is experimental (solid)
 Each data point on solid curves based on up to 40 algorithm executions (a.e.)

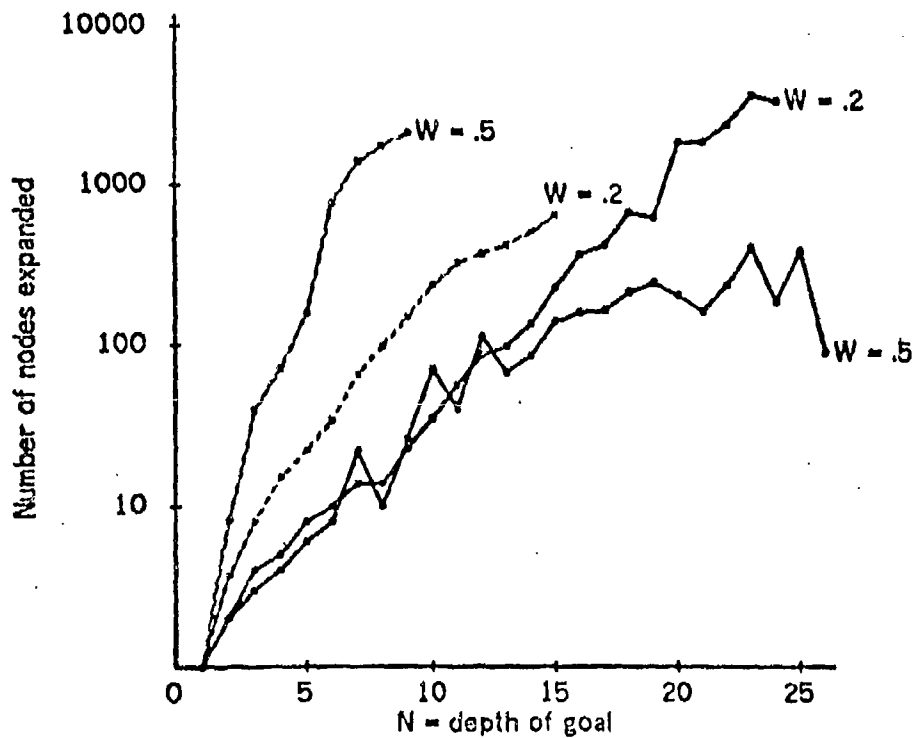


Figure 3.5-6 Analogous to Figure 3.5-2, for heuristic K_3
 $XWORST(K_3, W, N)$ is predicted (dash), $XMAX(K_3, W, N)$ is experimental (solid)
 Each data point on solid curves based on up to 40 algorithm executions (a.e.)

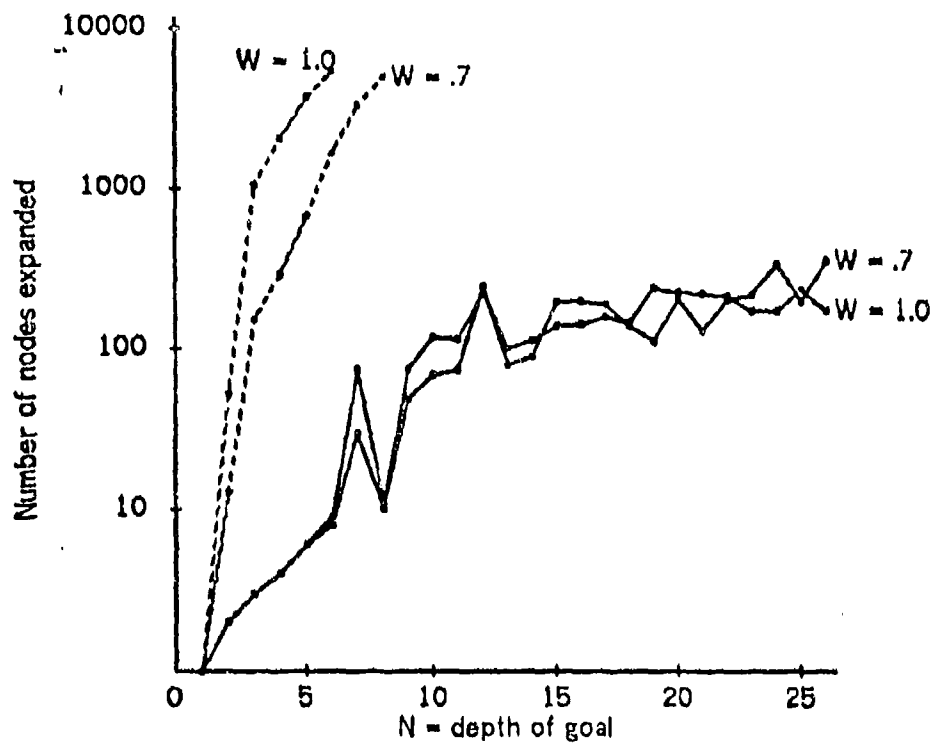


Figure 3.5-7 Analogous to Figure 3.5-3, for heuristic K_3
 $XWORST(K_3, W, N)$ is predicted (dash), $XMAX(K_3, W, N)$ is experimental (solid)
 Each data point on solid curves based on up to 40 algorithm executions (a.e.)

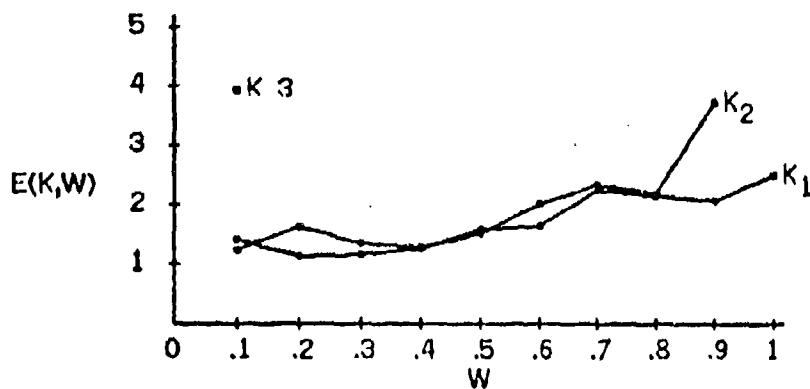


Figure 3.5-8 $E(K,W)$ = RMS of factor of difference between
 $XWORST(K, W, N)$ and $XMAX(K, W, N)$ averaged over common values of N
 Each data point represents up to 895 experimental observations

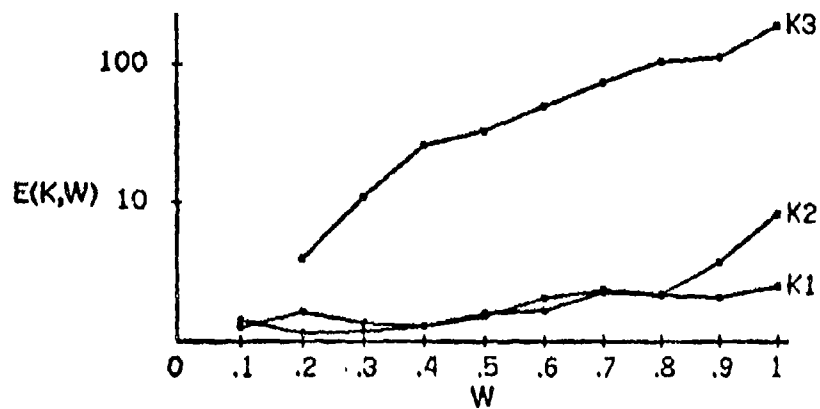
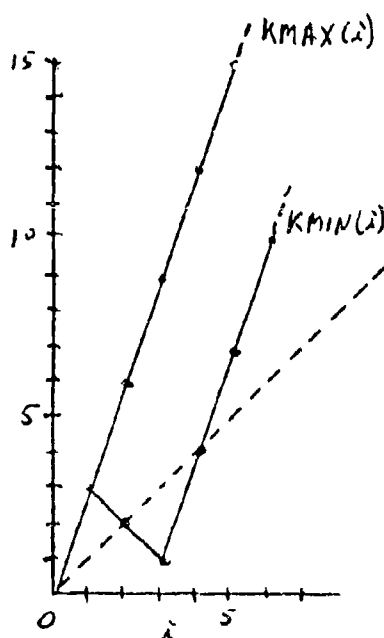


Figure 3.5-9 $E(K,W)$ = RMS factor of difference between $XWORST(K,W,N)$ and $XMAX(K,W,N)$
 (Different scale of ordinate axis from Figure 3.5-8)
 Experimental observations (for $XMAX$) based on more than 26,000 algorithm executions

4	5	4	3	4	3	2	3	2	3	2	3	4
5	4	3	4	3	2	3	2	3	2	3	2	3
4	5	4	3	2	3	4	1	2	1	4	3	2
5	4	3	4	3	2	1	2	3	2	1	2	3
4	5	4	3	2	3	2	3	0	3	2	3	2
5	4	3	4	3	2	1	2	3	2	1	2	3
				2	3	4	1	2	1	4	3	2
				3	2	3	2	3	2	3	2	3
				4	3	2	3	2	3	2	3	4

Figure 3.5-10 The "Knight's-Move" graph on an unbounded board (square on board = node in graph; an edge connects two nodes if the corresponding squares are separated by a knight's move). Numbers in squares indicate minimum number of knight's moves from square labelled "0" (i.e., distance in the graph between these two nodes).

Figure 3.5-11
Let $K_{\text{rect}}(s, t)$ denote the rectilinear distance function between squares s and t on the board. The diagram plots the bounding functions of K_{rect} when used as an estimator of the number of Knight moves required to move from square s to square t (i.e., as an estimator of distance between nodes in the Knight's-move graph).



CHAPTER 4

Experimental Case Studies of Backtrack vs. Waltz-type vs.
New Algorithms for Satisficing Assignment Problems¹

4.0 Summary of Chapter

Any instance of a certain class of satisficing assignment problem (or SAP, defined formally in Section 4.1.1, and as distinguished from optimizing assignment problems) can be solved using the so-called backtrack search algorithm, as defined in general form by Golomb & Baumert [1965], but little is known in general about the computational requirements of this algorithm. Believing the backtrack algorithm to be inefficient, Waltz [1972, 1975] and other researchers in artificial intelligence have devised an alternative general algorithm for SAPs that is based on constraint satisfaction principles, but claims about its efficiency have been based on skimpy hard data. Mackworth [1977] surveys and adds to reports by Sussman & McDermott [1972], Gaschnig [1974], and others documenting the inefficiencies of backtrack in specific instances, and Mackworth also surveys the generalizations of Waltz' algorithm given by Gaschnig [1974], Rosenfeld, et al., [1976], and others. Mackworth conjectures of the backtrack algorithm that "the time taken to find a solution tends to be exponential in the number of variables, both in worst case and on the average" [1977, p. 100], and that Waltz-type algorithms are "clearly more effective than automatic backtracking" [1977, p. 116]. Here we report experimental results that, among other things, contradict these conjectures in almost all cases observed.

The results we report compare experimental performance measurements under identical conditions of the backtrack algorithm, a new version of the Waltz algorithm called DEEB, and three new general algorithms for SAPs, which we call BACKMARK, BACKJUMP, and DEELEV. (BACKMARK and BACKJUMP do backtracking with less redundancy than the classic algorithm; DEELEV backtracks to a partial solution at level 1 in the search tree and then gives control to DEEB.) These algorithms are compared by each of three related performance measures, using as case studies four sample sets of SAPs:

- S1) a set of so-called N-Queens SAPs (i.e., place N queens on an NxN chess board so that no two attack each other); in this sample set there is one sample for each value of N, in which the "candidate values" (i.e., squares in a row of the board) of each "problem variable" (i.e., queen) are ordered in the "obvious" left to right order;

¹ The bulk of the experimental data reported in this chapter appeared first in [Gaschnig 1978] and [Gaschnig 1977b]. Algorithm BACKMARK was first described in [Gaschnig 1977b] (there called BKMARK). Algorithm BACKJUMP was first described in [Gaschnig 1978] (there called BKJUMP).

- S2) a set of N-queens SAPs in which for each value of N there are 30 to 100 samples in which the candidate values for each problem variable are first permuted randomly before commencing the search;
- S3) a set of randomly generated SAPs whose size and "degree of constraint" (L) parameters are chosen to match those of individual N-Queens SAPs (one parameter set for each value of N tested, 50 to 250 samples per distinct parameter set);
- S4) a set of randomly generated SAPs having identical size and varying values of L, hence suitable for determining how efficiency varies with degree of constraint, all other things being equal. (150 samples for each of 9 values of L)

The three performance measures for which we collected data are: T, the number of "pair-test"s, an instance of which for the 8-Queens problem tells whether a queen on a given square attacks a queen on another given square; D, the number of distinct pair-tests executed during a search; and $M = T/D$, a redundancy ratio measuring the number of times each distinct pair-test is computed. Mostly, we are concerned with the number of steps to find any solution as opposed to find all solutions. For sample sets S2, S3, and S4, we take the mean value of the performance measures over the samples corresponding to each value of N, and measure its accuracy in estimating the exact value to which it corresponds.

By comparing the performances of the four algorithms mentioned above for N-queens SAPs using "left-to-right" candidate value ordering (i.e., sample set S1) with the corresponding performances for N-Queens SAPs using random candidate value (c.v.) ordering (sample set S2), our purpose is to determine the extent to which randomizing the inputs to these algorithms in this way affects the resulting performance.

By comparing the performances of the algorithms for "random N-Queens" problems (sample set S3) with the corresponding performances for the N-queens problems using random c.v. ordering (sample set S2), our purpose is twofold: first, simply to generalize the comparative algorithm performance data from N-Queens SAPs to a broader class of SAPs, and second, to determine whether "natural" or "particular situation" problems such as N-Queens SAPs are typical of or distinguishable from parametrically similar problems that are generated randomly. The outcome suggests whether future analyses of simple i.i.d. probabilistic models of SAPS can yield accurate predictions of the algorithms' performances for "natural" problems, i.e., whether results for randomly generated problems permit accurate predictions for particular "structured" problems.

Note that as in Chapter 2, our technical objective here is simply to obtain experimentally the performance values plotted in the figures of this chapter and tabulated in Appendix C of the dissertation. Hence we provide a body of quantitative data against which to test future speculations and mathematical theories. The results of more than 13,000 distinct algorithm executions support, among others, the following conclusions:

- C1) In all observed cases of N-Queens SAPs (sample sets S1 and S2), the new algorithm BACKMARK executes fewer pair-tests ($T_f(N)$) than do algorithms BACKTRACK, DEEB, and BACKJUMP under identical conditions, in some cases fewer by a factor of 10. BACKMARK is observed to approach optimality more closely than the other three algorithms with

respect to the M_f redundancy ratio: few pair-tests are ever recomputed. To be concrete, BACKMARK finds solutions to the 50-Queens problem (having a search space of size about 10^{84}) at the average rate of one per 9 cpu-seconds on a DEC KL-10.

- C2) In almost all observed cases of N-Queens SAPs (sample sets S1 and S2), the Waltz-type algorithm DEEB executes more pair-tests on the average than do the other three algorithms under identical conditions.
- C3) For N-Queens SAPs (sample sets S1 and S2) and each algorithm, we also observe that randomizing the ordering of candidate values of each problem variable before commencing the search causes fewer pair-tests (mean $T_f(N)$) to be executed on the average for the larger values of N tested than if the "left-to-right" c.v. ordering is used, fewer by as much as a factor of 498 in one case observed! The difference in efficiency between random c.v. ordering and "left-to-right" c.v. ordering is much less for smaller values of N tested.
- C4) Conclusions C1 and C2 above are further supported by analogous data obtained for a set of "random-N-Queens" SAPs (sample set S3). Comparison of these "random-N-queens" data with the corresponding "N-Queens" data (sample set S2) shows that these two sample sets of SAPs are sharply more distinguishable for $N \geq 10$ than for $N < 10$ (i.e., the ratios of corresponding algorithm performances differs sharply from the value 1), and are sharply less distinguishable by algorithm DEEB than by the other three algorithms. Note in particular that for $N \geq 10$, N-Queens SAPs require many more pair-tests to be executed on the average than is the case for the corresponding random-N-Queens SAPs. (Why this should be the case remains unknown.)
- C5) Results reported for sample set S4 also support conclusions C1 and C2. Furthermore, the results indicate that the number of steps executed (mean $T_f(L)$) depends strongly on degree of constraint (L), spanning a range whose extremes differ by a factor of 791 among the cases tested! For each of the four algorithms tested, a plot of the data suggests the existence of a single sharp peak in $T_f(L)$ at $L \sim 0.6$. The peak and range of performance are clearly reflected in both $D_f(L)$ and $M_f(L)$ as well as in $T_f(L)$. These data show in particular that the Waltz-type algorithm does not better the others on the highly constrained problems tested.

These specific experimental results contradict some previous conjectures, support others, and reveal new phenomena. They support the more general statement that relatively simple algorithms applied to relatively simple problems can yield rather complex behavior. The present data expose trends in efficiency and other patterns of performance that vary with the algorithm, the set of problem instances, and the performance measure. Seeing these raw performance data and recognizing these trends and patterns therein helps to increase our understanding of the complex behaviors of these algorithms, providing a more extensive basis in fact for future generalizations.

We make no claims about the performances of these algorithms except for the cases tested here, but propose additional experiments to provide evidence on which to base such claims.

All good intellects have repeated, since Bacon's time, that there can be no real knowledge but that which is based on observed facts.

Auguste Comte

4.1 Backtrack vs. Waltz-type Algorithms: What to Measure and Why

4.1.1 Definitions, Examples, and Elementary Results

In this chapter we measure experimentally the performances of several algorithms -- some known previously and some new -- in solving selections of instances of a certain class of satisficing assignment problems, of which the Eight Queens problem and map coloring problems are elementary examples.

The problems spanned by this class differ greatly in their individual characteristics. This makes the backtracking algorithm and the other algorithms considered here very useful, in that they can be applied in many diverse circumstances. On the other hand, the generality of the problem class also makes a general analysis of the algorithms problematic, with the result that it is difficult to predict the performances of the algorithms in detail for a particular problem to be solved:

"One of the chief difficulties associated with the so-called backtracking technique for combinatorial problems has been our inability to predict the efficiency of a given algorithm, or to compare the efficiencies of different approaches, without actually writing and running the programs." [Knuth 1975, p. 121]

Knuth [1975] derived analytically a predictor for the number of steps executed by BACKTRACK for the case in which the solution criterion is to find all solutions of the given problem. Here we are mostly concerned with the case of finding any one solution, which would seem to be more realistic in many practical circumstances than finding all solutions, and which also seems to be more difficult to analyze mathematically. Accordingly, here we eschew any mathematical analysis of these algorithms, instead choosing simply to measure the algorithms' performances under diverse experimental conditions. Our objective then is to measure in case studies the values against which a future analysis could test its predictions, and to reveal patterns in the performances that may provide insight and direction by which to guide the development of such analysis. In particular, we wish to subject to the test of

hard data previous conjectures about the relative performances of backtracking and Waltz' constraint satisfaction algorithm.

In this section we: define formally a class of satisficing assignment problems; cite and contrast familiar instances of these problems; define a version of the classic backtrack algorithm and illustrate its inefficiency by an example; define performance measures by which to compare the algorithms considered in this chapter; describe the constraint satisfaction algorithm devised by Waltz as an alternative to backtracking and cite conjectures in the literature about its performance; and then, having finished the preliminaries, report elementary experimental comparisons of the backtrack algorithm with the Waltz-type algorithm for a sample of N-Queens problems.

A satisficing assignment problem (SAP) is defined by a set of problem variables, each of which can take on any of a given set of candidate values. The object is to find an assignment of candidate values to problem variables that satisfies a given set of constraint relations. The "Eight Queens" problem is a well-known SAP in which eight queens must be placed on a chess board so that no two queens can take each other. In general:

Definition 4.1² A satisficing assignment problem (SAP) is a tuple $\{N, R_1, R_2, \dots, R_N, P_{12}, P_{13}, \dots, P_{1N}, \dots, P_{N-1,N}\}$ such that:

- a) N is a positive integer (denoting the number of problem variables $x_1, x_2, \dots, x_N$).
- b) $R_1, R_2, \dots, R_N$ are arbitrary finite sets. (Associated with each problem variable x_i is a specified set of a priori possible candidate values $R_i = \{v_{i1}, v_{i2}, \dots, v_{ik_i}\}$.)
- c) An assignment $A = (v_1, v_2, \dots, v_N)$ of candidate values to problem variables is an element of the cross product $U = R_1 \times R_2 \times \dots \times R_N$. $A(i)$ denotes the i 'th component of A , and A_i denotes the partial assignment $(A(1), A(2), \dots, A(i))$, for $1 \leq i \leq N$.
- d) For every i and j such that $1 \leq i < j \leq N$, there is defined a constraint relation $P_{ij} \subseteq R_i \times R_j$. The relation P_{ij} is termed proper if and only if $P_{ij} \subset R_i \times R_j$. Otherwise P_{ij} is the universal relation $U_{ij} = R_i \times R_j$. (Any two problem variables are either mutually constraining, or are not.) We also assume the constraint between any two problem variables is mutual, i.e., that $P_{ij} = P_{ji}$ for all i and j .

² Our formalism is similar to that of Mackworth [1977, pp. 99-100] and Knuth [1975]. Unlike Mackworth, however, we find it more useful to define the P_{ij} constraints as relations rather than as predicates (for reasons seen useful in Section 4.4).

- e) An assignment $A \in U$ is a solution if and only if for every i and j such that $1 \leq i < j \leq N$, $(A(i), A(j)) \in P_{ij}$ (or in predicate form, $P_{ij}(A(i), A(j))$ is valid).

In the 8-queens SAP $N = 8$, the problem variable x_i corresponds to the i 'th queen, and the candidate values of each queen consist of the a priori legal squares on which that queen can be placed. Since in any solution the queens must occupy distinct rows of the board, we take R_i to consist of the eight squares in row i ($k_i = 8$, for $2 \leq i \leq 8$). For symmetry reasons, however, we restrict R_1 to consist of the leftmost four squares of row 1 (i.e., $k_1 = 4$).³ In predicate form, $P_{ij}(A(i), A(j))$ is valid for assignment A if queen i on square $A(i)$ does not attack queen j on square $A(j)$. The eight queens problem generalizes to the N -Queens problem, the object of which is to place N queens on an N by N board so that no two attack each other.

Another familiar SAP is the problem of coloring a map (i.e., a finite undirected graph) of N regions with c colors so that no two neighboring regions are assigned the same color (e.g., see [Nijenhuis & Wilf 1974]). In this case N equals the number of regions of the map (i.e., the number of nodes in the graph), and k_i equals c for each i and $R_i = \{v_1, v_2, \dots, v_c\}$ for each i , and P_{ij} for non-neighboring regions i and j is the universal relation $U_{ij} = R_i \times R_j$. For neighboring regions i and j , P_{ij} is $U_{ij} - \{(v_k, v_k) \mid 1 \leq k \leq c\}$.

Other problems that can be formulated as SAPs include labeling in a particular way each of the line segments in a two-dimensional projection of a scene of polyhedra [Waltz 1972, 1975], other computer vision applications cited by Mackworth [1977, pp.115-116], finding Euler circuits or Hamiltonian circuits or spanning trees of a graph [Nijenhuis & Wilf 1975], cryptarithmic [Simon 1969], [Newell & Simon 1972], [Gaschnig 1974], the Instant Insanity puzzle [Knuth 1975], [Brown 1968], the SOMA cube puzzle [Fillmore & Williamson 1974, p. 51], [Gardner 1972], and space planning problems [Eastman 1972, 1974]. Other examples of SAPs are cited in [Golomb & Baumert 1965], [Lauriere 1978], and [Mackworth 1977]. The N -Queens SAPs are considered in [Floyd 1967], [Fikes 1970], and [Mackworth

³ Note that additional symmetry considerations might suggest an alternative SAP formulation of this problem that may have fewer assignments that are solutions. In general, such symmetry issues are beyond the scope of this dissertation; here all statements concerning SAPs assume some definite formulation of the form given in Definition 4-1. The formulation of the 8-queens SAP stated above is the one used in these experiments. In our formulation of the N -Queens problems, $k_1 = \lceil N/2 \rceil$ and $k_2 = k_3 = \dots = k_N = N$.

1977].⁴

Note that the definition of SAP can be generalized from having binary constraint relations $P_{ij} \subseteq R_i \times R_j$ to ternary constraint relations $P_{ijm} \subseteq R_i \times R_j \times R_m$ to r -ary constraint relations $P_{i_1 i_2 \dots i_r} \subseteq R_{i_1} \times R_{i_2} \times \dots \times R_{i_r}$, where $1 \leq r \leq N$. Such cases might be distinguished by the names 2-SAP, 3-SAP, and r -SAP, respectively. For example, the cryptarithmic problems considered in [Gaschnig 1974] are 3-SAPs, but happen to reduce to 2-SAPs because of ordering considerations. Here we consider only 2-SAPs.⁵

Mackworth [1977] describes another general criterion by which SAPs may be distinguished. Mackworth associates with each SAP a labeled directed graph whose N nodes correspond (one-to-one onto) with the N problem variables, and directed edges e_{ij} and e_{ji} connect the nodes corresponding to problem variables x_i and x_j if and only if P_{ij} is proper. Hence a SAP having N problem variables in which every P_{ij} is proper corresponds to a complete graph on N nodes. Here we say that such a SAP has a complete consistency graph; otherwise a SAP is said to have an incomplete consistency graph. Among the SAPs cited above, N -Queens, Instant Insanity, and Soma cube SAPs have complete consistency graphs (e.g., every queen (problem variable) imposes a constraint directly on the placement of every other queen). In contrast, Waltz' line drawing problem, map coloring, and the space planning SAPs have incomplete consistency graphs (e.g., each junction (problem variable) in one of Waltz' line drawing problems is connected to only a few other junctions, but may constrain the assignments of other junctions indirectly by a propagation of constraint along a path of connected junctions). Except for the results for map coloring problems reported in Section 4.5.1, the present experiments assume exclusively SAPs having complete consistency graphs.

⁴ In addition, there is a growing body of results concerning related problems. So called "relaxation" methods (probabilistic versions of the Waltz constraint satisfaction algorithm) have been applied to problems that are probabilistic analogues of SAPs [Rosenfeld et al. 1976], [Zucker 1976], [Zucker et al. 1976]. In other settings, Sussman and Stallman [1977] have devised a type of "dependency-directed backtracking", and Lindstrom ([1977], [1978]) has devised versions of "non-forgetful backtracking."

⁵ Freuder [1978] proposes an iterative algorithm that solves an r -SAP by first solving a related $(r-1)$ -SAP, but more work is needed in analyzing the computational efficiency of this algorithm.

To be concrete, the version of the backtrack algorithm used in these experiments is defined below as a SAIL procedure⁶, in a form that halts after finding a single solution (if any exist). Procedure BACKTRACK applies to any SAP having N problem variables, each problem variable x_i having $k[i]$ a priori possible candidate values. PAIRTEST is an external procedure whose definition is problem dependent, such that an assignment vector $A[1:N]$ of candidate values to problem variables is a solution if and only if $\text{PAIRTEST}(i, A[i], j, A[j])$ is true for all $1 \leq i < j \leq N$ (e.g., in the 8-Queens problem, if and only if no queen can take any other queen). Note that vector A in procedure BACKTRACK contains indices to the actual candidate values, which are encoded in an externally defined array $\text{VALUES}[1:N, 1:k_{\max}]$, where $k_{\max}$ is the maximum of $k_1, k_2, \dots, k_N$. Hence the arguments to PAIRTEST specify two elements of VALUES. Top level invocation for the 8-queens problem takes the form

$\text{tmp} \leftarrow \text{BACKTRACK}(1, 8, A, B),$

where the actual parameters $A[1:8]$ and $B[1:8]$ are one dimensional arrays. In general, arrays A and B have dimensionality $A[1:N]$ and $B[1:N]$. Initial values of A are irrelevant; the initial value of $B[i]$ is assumed to be k_i . BACKTRACK returns -1, with A containing the indices of the candidate values constituting the first solution found, or returns 0 if no solution exists. The version of BACKTRACK used in the present experiments to find all solutions is a trivial variation of the procedure given below. For brevity, the symbol "!" stands below for "comment". The code below is valid for any 2-SAP, given suitable actual parameters N and the $k[i]$ and a problem dependent definition for procedure PAIRTEST.

⁶ The SAIL language, an extension of Algol, is described in [Swinehart & Sproull 1971].

```

recursive integer procedure backtrack(integer var, n;
                                   integer array a, k);
! n = no. of problem variables;
! k[i] = no. of candidate values of problem variable i;
! var is the problem variable instantiated during current invocation;
begin
integer i, val;
boolean testflg;
for val ← 1 step 1 until k[var] do
    ! instantiate candidate values of var, one by one;
begin
testflg ← true;
for i ← new[var] step 1 while i < var and testflg do
    testflg ← pairtest(i, a[i], var, val);
    ! does new queen on this square attack queen i on its assigned square?;
if testflg then
    ! if passed all tests, then...;
begin
a[var] ← val;
if var = n then return(-1)
else if backtrack(var+1, n, a, k) = -1 then return(-1)
    ! found solution, so unwind to outermost call;
    ! if son says to unwind, then tell father to unwind;
end
end;
return(0)
    ! backtrack and continue search;
end;

```

We define an elemental unit of computation to be a consistency test or pair-test, which in the case of N-queens problems tells whether a queen on one specified square attacks a queen on another specified square. Formally, a pair-test is identified by a 4-tuple (i, y, j, z) , where $1 \leq i < j \leq N$ and $y \in R_i$ and $z \in R_j$, which determines whether $(y, z) \in P_{ij}$. (In predicate form, the unit is an execution of $P_{ij}(A(i), A(j))$.) In the experiments, a pair-test corresponds to a single invocation of procedure PAIRTEST(i, u, j, v), where in this context u and v are indices to the elements of R_i and R_j respectively under the ordering in which these sets are input (where $1 \leq u \leq k_i$ and $1 \leq v \leq k_j$).

The numerous distinct partial instantiations of problem variables form a tree, as depicted by the following incomplete trace of BACKTRACK for the 8-queens problem. Each occurrence of "T" and "F" in the trace indicates the outcome of a single pair-test. For example, the entry "6,4 TTTF" in portion "A" indicates that the pair-tests with arguments (6,4,1,1), (6,4,2,1), (6,4,3,5), and (6,4,4,2) returned the values True, True, True, and False,

respectively. Since a queen on square (6,4) attacks a queen on square (4,2), this instantiation of problem variable 6 fails to be consistent with the particular assignments of the first four queens, and hence PAIRTEST(6,4,5,4) is not executed. This trace will also be referred to in Section 4.2 to motivate and demonstrate algorithms BACKMARK and BACKJUMP.

Incomplete trace of algorithm BACKTRACK for 8-Queens SAP:
 (Labels below on the right are referred to in the text.)

1,1 x,y = queen x on square at row x, column y

```

2,1 F
2,2 F
2,3 T
  3,1 F
  3,2 TF
  3,3 F
  3,4 TF
  3,5 TT
    4,1 F
    4,2 TTT
      5,1 F
      5,2 TTTF
      5,3 TF
      5,4 TTTT
        6,1 F
        6,2 TTF
        6,3 TF
        6,4 TTTF
        6,5 TTF
        6,6 F
        6,7 TF
        6,8 TTF
          5,5 F
          5,6 TF
          5,7 TTF
          5,8 TTTT
            6,1 F
            6,2 TTF
            6,3 TF
            6,4 TTTF
            6,5 TTF
            6,6 F
            6,7 TF
            6,8 TTF
              4,3 TF
              4,4 F
              4,5 TF
              4,6 TTF
              4,7 TTT
                5,1 F
                5,2 TTTT
                  6,1 F
                  6,2 TTF
                  6,3 TF
                  6,4 TTTTT
                    7,1 F
                    7,2 TTTTF
  
```

Labels on the right:

- C1 (bracketed next to 5,1 F through 5,4 TTTT)
- A (bracketed next to 6,1 F through 6,8 TTF)
- C2 (bracketed next to 5,5 F through 5,8 TTTT)
- B (bracketed next to 6,1 F through 6,8 TTF)
- F (bracketed next to 4,3 TF through 4,7 TTT)
- D (bracketed next to 5,1 F through 5,2 TTTT)
- E (bracketed next to 6,1 F through 6,4 TTTTT)

Algorithms for SAPs may be redundant in the sense that some pair-tests may be

executed more than once. Indeed, measuring the extent of such redundancy is a major focus of the present experiments. Accordingly, we distinguish three related performance measures:⁷

T = total number of pair-tests executed by an algorithm B for a SAP S

D = number of distinct pair-tests executed under the same conditions

$M = T / D$

So $M = 1$ if and only if all pair-tests executed are distinct, i.e., none are recomputed. We refer to M as the redundancy ratio. To illustrate our performance measures, for the portion of the execution traced above $T = 107$, $D = 75$, and $M = 107/75 = 1.43$.

For SAPs that have at least one solution, the minimum number of pair-tests executed by BACKTRACK is $N(N-1)/2$, which is observed if and only if the assignment consisting of the first candidate value (c.v.) of each problem variable happens to be a solution. In this case, each c.v. of the assignment is "pair-tested" against every other, and no other pair-tests are executed. Intuition suggests that any algorithm "valid" for arbitrary SAPs must execute at least $N(N-1)/2$ pair-tests (given a SAP having N problem variables and having at least one solution), the argument being that this is the number of pair-tests required to verify that a given assignment is a solution if in fact it is. In Section 4.5.4 we formalize this notion by defining abstractly the set of all valid algorithms for SAPs, then proving by an adversary argument that any algorithm in this class (including all algorithms considered in the present experiments) must execute at least

$$T_{\min}(N) = N(N-1)/2$$

pair-tests in solving an arbitrary SAP having N problem variables and having a solution, or otherwise be invalid.⁸

For a given SAP, the total number of possible distinct pair-tests is determined by the values of N and the k_i , thus:

⁷ Note that the performance measure here called M was called D in [Gaschnig 1977b].

⁸ The formal proof of this is ancillary to the present experiments, and hence we defer it to Section 4.5.4. In the figures presented in the intervening Sections, we simply plot the values of $T_{\min}(N) = N(N-1)/2$ along with experimental data for comparison sake.

$$D_{\max}(N, k_1, \dots, k_N) = \sum_{1 \leq i \leq N-1} \sum_{i+1 \leq j \leq N} k_i * k_j \quad (\text{for SAPs in general})$$

$$= (N-1) N \lceil N/2 \rceil + N^2 (N-1)(N-2)/2 = O(N^4) \quad (\text{for N-Queens SAPs})$$

$D_{\min}(N, k_1, k_2, \dots, k_N) = T_{\min}(N)$ always, since the pair-tests counted by $T_{\min}(N)$ are distinct. A measure of the size of the search space is SAS, the total number of distinct possible assignments⁹, defined thus:

$$SAS(N, k_1, \dots, k_N) = \prod_{1 \leq i \leq N} k_i \quad (\text{for SAPs in general})$$

$$= O(N^N) \quad (\text{for N-Queens SAPs})$$

For N-Queens SAPs we write $D_{\max}(N)$ and $SAS(N)$, since they can be written as functions only of N . For sake of comparison, the values of $T_{\min}(N)$, $D_{\max}(N)$, and $SAS(N)$ for N-Queens SAPs are plotted as functions of N in some of the figures of this chapter. T_f , D_f , and M_f denote values observed when the solution criteria is to find any solution (i.e., a first solution); T_a , D_a , and M_a similarly denote values observed when the criteria is to find all solutions.

Procedures of the sort devised by Waltz [1972, 1975]^{10,11} operate on principles different from those employed by BACKTRACK: instead of instantiating cases as does BACKTRACK, they eliminate a candidate value $z \in R_i$ of problem variable x_i when it is detected for some other problem variable x_j that

$$\nexists y \in R_j \text{ such that } (z, y) \in P_{ij} \quad (4.1.1-1)$$

⁹ SAS is mnemonic for "size of assignment space".

¹⁰ A multitude of names have been suggested for Waltz-type algorithms. Mackworth suggests "network consistency" algorithms, but in a sense any algorithm valid for SAPs, including BACKTRACK, can be termed a network consistency algorithm, since the algorithm finds an assignment satisfying the consistency graph or network. Rosenfeld, et al. [1976], Zucker [1976] and Zucker, et al. [1976] and others name their probabilistic version a "relaxation" method. Winston [1977] prefers "range constriction". The name used here, "Domain Element Elimination" algorithm, seems more descriptive. The version of the algorithm used in the present experiments is called DEER, an acronym for "Domain Element Elimination with Backtracking".

¹¹ Mathematical analyses of such procedures for SAPs can be found in [Montanari 1974], [Mackworth 1977], [Freuder 1978], [Haralick & Shapiro 1979a], and [Haralick & Shapiro 1979b].

Under this condition no assignment A in which $A(i) = z$ can be a solution. The elimination of candidate value z of problem variable x_i can in turn cause a candidate value w of another problem variable x_k to be eliminated in like manner, if z was the only element providing consistency with w under relation P_{ik} . The propagation of inconsistency can continue throughout the whole consistency graph, as demonstrated explicitly by Waltz and in [Gaschnig 1974] and [Mackworth 1977].

Waltz-type algorithms halt under one of three conditions. One halting condition occurs if the domain of each problem variable is reduced in the above manner to a singleton (in which case a unique solution has been found). A second halting condition occurs if the domain of some problem variable is reduced to the null set, in which case the problem has no solution. In general, however, the algorithm halts after all constraints have been fully propagated, achieving the condition Mackworth [1977] calls "arc consistency". In effect, a Waltz-type algorithm reduces a given SAP S to another SAP S' such that $R'_i \subseteq R_i$ and $P'_{ij} \subseteq P_{ij}$ for all i and j , and such that an assignment A is a solution for S' if and only if A is a solution for S . This implies that $k'_i \leq k_i$ for all i , and hence $0 \leq SAS' \leq SAS$. In the case that $SAS' > 1$, the reduced problem S' may have 0, 1, or several solutions. Hence the basic Waltz algorithm can halt without having found a solution, if solutions exist, and can halt without having determined that no solutions exist, if none exist.¹²

This deficiency can be overcome by introducing additional constraint into the subproblem S' , when $SAS' > 1$. An obvious way of introducing additional constraint is to instantiate one of the problem variables x_i to one of its k'_i candidate values. The basic Waltz algorithm can then be executed again, resulting in a new subproblem. Additional problem variables may be instantiated and the basic Waltz algorithm applied until either a solution or a contradiction is discovered, after which one may backtrack and re-instantiate a problem variable to another candidate value. Hence the general method is to execute a backtrack search in which the basic Waltz procedure is executed at every node of the search tree produced by backtracking before the problem variable corresponding to that node is

¹² The basic Waltz algorithm described above has been formulated in similar ways by Gaschnig [1974] (there called CS-1), Mackworth [1977] (there called an arc consistency algorithm AC-3), and others. The variation used here is called DEE-0. Section 4.2.3 presents additional analysis of DEE-0, proving that the version used in the present experiments avoids certain minor inefficiencies arising from implementation details that Mackworth [1977, p. 114] observed to be present in CS1.

instantiated to any of its candidate values. The Waltz-type algorithm with backtracking used in these experiments, called DEEB, is functionally equivalent to and somewhat more efficient than algorithm CS2 defined in Gaschnig [1974].

Given the preceding definitions, we may now attempt to address precisely Mackworth's intent in conjecturing of backtrack algorithms that "...the time taken to find a solution tends to be exponential in the number of variables, both in worst case and on the average" [Mackworth 1977, p. 116]. Mackworth does not specify whether "time" means cpu-time, number of pair-tests, or some other measure. Here we shall presume number of pair-tests, and we present evidence subsequently that T and cpu-time are approximately linearly related. Presumably, by "find a solution" Mackworth intends T_f as opposed to T_a , and the phrase "exponential in the number of variables" suggests $T_f(N)$. Defining "both in worst case and on the average" depends on defining a set of problem instances over which such measurements are to be aggregated.

Constituting what is to the best of our knowledge the first experimental evidence against which to test Mackworth's conjecture that Waltz-type algorithms are "clearly more effective" (i.e., execute fewer pair-tests) than the backtrack algorithm under identical conditions (including identical problem instances and identical performance measures), Figure 4.1.1-1 compares BACKTRACK with DEEB by $T_f(N)$ for N-Queens SAPs, $N = 4, 5, \dots, 17$. The following tabulation compares BACKTRACK with DEEB by $T_a(N)$:

N:	4	5	6	7	8	9
BACKTRACK	42	236	1808	5345	23376	136807
DEEB	78	379	1832	4218	14118	68239

Hence these data support Mackworth's conjecture for the larger values of N tested, but not for the smaller values of N tested. More extensive comparative data are given in Sections 4.3 and 4.4.

To show the relation between total number of pair-tests executed and the number of distinct pair-tests executed, both for the case of finding one solution and the case of finding all solutions, Figure 4.1.1-2 plots $T_f(N)$, $D_f(N)$, $T_a(N)$ and $D_a(N)$ for BACKTRACK. Inspection of this figure suggests that the plotted values of $T_a(N)$ can be closely approximated by a function that grows exponentially with N . The values of $T_a(N) / T_a(N-1)$ for $N = 5, 6, 7, 8, 9$ are 5.619, 4.271, 5.303, 4.373, and 5.852 respectively, indicating a possible lower order even-odd pattern in the growth of the values of $T_a(N)$ with N . Note in Figure 4.1.1-2 that

$D_a(N) / D_{\max}(N)$ approaches the value 1 as N increases. For $N = 4, 5, \dots, 9$, these values are .514, .695, .736, .920, .959, .995, respectively.

Figure 4.1.1-3 plots the redundancy ratios $M_f(N)$ and $M_a(N)$ based on the data in Figure 4.1.1-2. These data show that the redundancy of BACKTRACK grows sharply with increasing size of the problem. Again, Sections 4.3 and 4.4 report much more extensive data. Perusal of such $M_f(N)$ data in fact motivated an attempt to define a backtrack-like algorithm (namely BACKMARK) that eliminates much of this redundancy. Here then is a case in which performance measurement experiments yielded insights that led to a new algorithm.

To determine whether the number of pair-tests, T_f , accurately indicates the total cpu-time required, cpu-time measurements were recorded for the BACKTRACK executions to find first solution plotted in Figure 4.1.1-1. The results suggest a linear relation between $T_f(N)$ and $\text{cpu}_f(N)$. For $\text{cpu}_f(N)$ given in milliseconds, the experimental data indicate the formula

$$T_f(N) \sim 15 \text{ cpu}_f(N)$$

where the coefficient is observed to be as small as 13.04 for $N = 4$ and ranges between 15.09 and 15.66 for $8 \leq N \leq 17$, being generally (but not always) larger when $T_f(N)$ is larger. The absolute value of this coefficient is not interesting, since its value depends on the particular machine used to execute the algorithm, and indeed to a lesser extent on the machine's load factor at the time of execution (this is true at least for the DEC KL-10). However, the small spread of the coefficient value with N indicates that whatever the machine and load factor, the values of $T_f(N)$ and $\text{cpu}_f(N)$ differ, to a first approximation, only by a multiplicative factor.¹³

Of course, whether this simple multiplicative relation between T_f and cpu-seconds holds for problems other than the N -Queens remains to be determined by further experiment (see Section 4.1.2 for additional data). In general, the quantity $\text{cpu}(N)$ is the sum of two quantities, namely the cumulative cpu-time required to execute the numerous invocations of the PAIRTEST procedure itself (call this problem dependent term $\text{cpu-pd}(N)$), and the cumulative cpu time for executing the BACKTRACK procedure excluding the PAIRTEST invocation (problem independent term $\text{cpu-pi}(N)$).

¹³ Indeed, if the values of $\text{cpu}_f(N)$ and $T_f(N)$ are superimposed in a single plot and scaled independently on the ordinate axis so as to register as closely as possible, the two curves match to the eye almost exactly.

4.1.2 "Obvious" vs. Random Orderings of Candidate Values

The data plotted in the figures of Section 4.1.1 represent a very special case: a single set of related SAPs (i.e., N-Queens SAPs as opposed to all other SAPs), a particular order of instantiating the N problem variables and, for each problem variable, a particular order of instantiating the candidate values of that problem variable. This section reports how the observed values of the performance measures vary with the choice of candidate value ordering.

In the experiments of Figures 4.1.1-1, 4.1.1-2, and 4.1.1-3 the candidate squares for each queen comprise a row of the board, and the squares are ordered from left to right ("left-to-right" candidate value (c.v.) ordering). In fact, for each queen $8!$ distinct orderings are possible, corresponding to the permutations of the squares. In general, for each problem variable x_i , there are $k_i!$ distinct orderings of the k_i candidate values. For a given SAP, this gives a total of $\prod_{1 \leq i \leq N} k_i!$ distinct c.v. orderings.

Note in Figure 4.1.1-1 the "zig-zag" pattern of growth, the particular rate of growth in $T_f(N)$ for each algorithm, and the relative efficiencies of the two algorithms. Our objective now is to determine the extent to which these observed characteristics can be attributed to the choice of "left-to-right" c.v. ordering, and to what extent to the inherent structure of the N-queens problems. Toward this end, Figure 4.1.2 1 plots measurements of $T_f(N)$ assuming the ordering of the candidate squares for each problem variable is chosen randomly. Figure 4.1.2-1 shows the mean, maximum, and minimum values of $T_f(N)$ observed over a set of $m(N)$ samples of the N-Queens problem. The order of instantiating the candidate values of each problem variable in each problem instance in the sample is chosen independently, with replacement, and uniformly such that each of the $\prod_{1 \leq i \leq N} k_i!$ c.v. orderings is equally likely. For $N \leq 7$, $m(N) = 30$; $m(N) = 70$ for $8 \leq N \leq 14$; and $m(N) = 100$ for $N \geq 15$. Using these values of $m(N)$, the value of the sample standard deviation of the sample mean of $T_f(N)$ is observed to range from 6 to 17 percent of the value of the sample mean.¹⁴ The vertical bars in the figure depict an interval of two sample standard deviations of the sample mean of $T_f(N)$ ---

the computation is the same as described in Chapter 2 for XMEAN, LMEAN, and so on.

Figure 4.1.2-1 makes visually apparent the difference in effect between left-to-right candidate value ordering and random candidate value ordering. Let $T_{f|obv}(N)$ denote the $T_f(N)$ data plotted in Figure 4.1.1-1, and let $T_{f|ran}(N)$ denote the mean value of $T_f(N)$ plotted in Figure 4.1.2-1. The data of Figure 4.1.2-1 indicate that $T_{f|obv}(N) \sim T_{f|ran}(N)$ over the range $4 \leq N \leq 13$, and that $T_{f|obv}(N) \gg T_{f|ran}(N)$ for $N > 13$. Note also that the even-odd pattern in $T_f(N)$ observed in Figure 4.1.1-1 is greatly reduced for random c.v. ordering.

Note in Figure 4.1.2-1 the very large difference between the minimum and maximum values of $T_f(N)$ for random c.v. ordering. For $10 \leq N \leq 15$, the maximum exceeds the minimum by more than a factor of 100. Note also that the minimum $T_f(N)$ value observed for random c.v. ordering is close to $T_{min}(N)$. The ratio of the former to the latter ranges from 1.6 to 3.55 for these data. These data suggest future efforts to devise heuristics for choosing c.v. orderings in such a way as to minimize T_f .

To determine whether the observed differences between $T_{f|obv}(N)$ and $T_{f|ran}(N)$ reflect differences in $D_f(N)$, or in $M_f(N) = T_f(N) / D_f(N)$, or in both, we measured the mean values of $D_{f|ran}(N)$, plotted in Figure 4.1.2-2, and $M_f(N)$, in Figure 4.1.2-3, over the N-Queens sample set using random c.v. ordering. The data indicate that for each of the three performance measures, both types of candidate value orderings give comparable values for $4 \leq N \leq 13$, whereas "left-to-right" c.v. ordering gives much larger values than random c.v. ordering for $N > 13$. Why this should be the case remains an open question. Such data demonstrate explicitly the risk in extrapolating experimental results to values of an experimental parameter larger than those observed. Subsequent experiments of a different character (Section 4.4.2) also reveal striking difference in performance between values of N less than a certain threshold and those exceeding the threshold. The threshold of N observed in the latter case is different from that observed in the data of this section.

In the same experiment that measured the $T_a(N)$ values plotted in Figure 5.1.1-1, we

¹⁴ For each value of N , the value of $m(N)$ was chosen large enough so that this measure would be small. That is, a preliminary value of $m(N)$ was chosen and the experiment run and the value of the sample standard deviation of the sample mean of $T_f(N)$ was computed. If this value was observed to be greater than .2 times the sample mean of $T_f(N)$, then the experiment was run again with a larger value of $m(N)$.

determined (by counting during the search) the number of solutions, $\text{sol}(N)$, that exist for the N -queens problem, for $N = 4, 5, \dots, 9$, and the value of $\text{sol}(10)$ was determined using algorithm BACKMARK (see Section 4.2). These values are 1, 6, 2, 23, 46, 203, 362, respectively, i.e., growing monotonically with N with one exception. The mean values of $T_f(N)$ plotted in Figure 4.1.2-1 also exhibit exceptions to monotonicity. Note in particular the following relations:

$$\text{sol}(4) < \text{sol}(5)$$

$$\text{sol}(5) > \text{sol}(6)$$

$$\text{sol}(6) < \text{sol}(7)$$

$$T_f(4) > T_f(5)$$

$$T_f(5) < T_f(6)$$

$$T_f(6) > T_f(7)$$

These values suggest the possibility of a simple relation between the number of solutions and the cost of finding a solution (i.e., first solution). Intuition suggest that the more solutions a SAP has, the more quickly one may be able to find one, all other things being equal. To investigate this possibility, Figure 4.1.2-4 plots the values of $\text{prod}(N) = T_f(N) * \text{sol}(N)$ derived from the previously cited data. Inspection suggests that the observed values of $\text{prod}(N)$ can be closely approximated by a function that grows exponentially with N . The values of $\text{prod}(N) / \text{prod}(N-1)$ for $N = 5, 6, \dots, 10$ are 5.817, 4.247, 5.479, 6.712, 6.542, and 5.435, respectively. These data suggest the possible existence of a first order exponential relation between the cost of finding a solution (i.e., $T_f(N)$) and the number of solutions. Designs for additional experiments to investigate this relation are described under item E4-7 in Appendix B.¹⁵

The cpu-time measurements described in Section 4.1.1 for the case of "left-to-right" c.v. ordering were recorded for this experiment also. The empirical relation $T_f(N) \sim 15 \text{ cpu}_f(N)$ is observed to hold for random c.v. ordering as well as for "left-to-right" c.v. ordering, where these are mean values over $m(N)$ samples, and $\text{cpu}_f(N)$ is in milliseconds as before. In this case, the coefficient value is as small as 12.97 for $n = 4$, and ranges between 15.23 and 15.79 for $8 \leq N \leq 15$, increasing monotonically with N over the range $4 \leq N \leq 15$.

BACKTRACK using random candidate value ordering is an instance of an algorithm that randomizes its inputs, as discussed by Weide [1977, pp. 304-305] from the point of view of analysis of algorithms research. It is known, for example, that Quicksort performs better if

¹⁵ Chapter 1, Section 1.5 ("Tradeoffs") under the heading "Level of detail" cites a rationale for not pursuing such experiments in the present dissertation.

the file to be sorted is random than if other assumptions hold [Sedgewick 1975]. Comparative data for the three other tested algorithms are given in Section 4.3. Hence the present experiments contribute to this facet of analysis of algorithms research.

4.1.3 Analysis of Waltz' Experimental Results

Although the poor efficiency of backtracking in solving line drawing assignment problems led Waltz to devise a new algorithm, he did not report explicitly many performance measurements of his algorithm. Waltz did report cpu-time measurements for six line drawings [Waltz 1972, pp. 21-22] or [Waltz 1975, p. 24]. Figure 4.1.3-1 plots these six cpu-time measurements against the corresponding size of the problem, as measured by the number of junctions in the line drawing. (The symbols W1, W2,..., W6 identify the line drawings in left to right, top to bottom order as they appear in the aforementioned sources.) Little can be concluded with confidence from so few data points. Winston however states:

"...This suggests that if the [Waltz] theory [of line drawings] succeeds in doing analysis [of line drawings], it should succeed in time proportional to the size of the scene. Fortunately, experiment verifies both success and proportional time." [Winston 1977, p. 69]

We can obtain additional insight about the relation between cpu-time and the size of the line drawing problem indirectly by determining the relation between the size of the assignment space (SAS) and the number of junctions N , and then between cpu-time and SAS, as follows. Figure 4.1.3-2 plots SAS against number of junctions for the same 6 line drawings. The calculation of the SAS values plotted in this figure is given below in the table below, in which rows represent junction types and columns represent line drawing diagrams. The table entry "7+4" for junction type "L" and figure W2, for example, indicates that line drawing W2 contains 11 junctions of type "L", four of which border the background of the scene, and seven of which lie in the interior of the line drawing. Description of junction types is given in [Waltz 1972, p. 15] or [Waltz 1975, p. 22]. The numbers in the "interior" and "background" columns of the table indicate the number of physically possible labelings of an instance of the indicated junction type, the two cases distinguishing the location of the vertex in the scene. These k_i values are taken from [Waltz 1972, p. 141] or [Waltz 1975, p. 51]. The SAS value for each line drawing is the product of the numbers of physically possible labelings of its vertices.

	W1	W2	W3	W4	W5	W6	interior	background
L	0+4	7+4	2+5	3+8	4+11	7+9	92	16
Arrow	1+3	0+2	0+7	1+7	2+9	1+7	86	12
T	0+1	4+1	0+5	1+7	1+9	5+7	623	96
Fork	1+0	1+0	2+0	3+0	5+1	4+0	826	26
Peak		2+1			0+2	0+2	10	2
K				1+0	1+1	1+1	213	2
X			0+1	1+1	1+1	1+3	435	72
Multi					0+1		128	8
KA				1+0	1+0		20	0
total	2+8	14+8	4+18	11+23	15+35	19+29		
SAS	1.2×10^{15}	2.5×10^{40}	3.9×10^{25}	3.0×10^{59}	3.7×10^{82}	8.4×10^{86}		

Figure 4.1.3-2 suggests that $SAS(N)$ is closely approximated by a function that grows exponentially with N . If this is true, and if $cpu(N)$ is closely approximated by a linear function, then $cpu(\log(SAS(N)))$ should also be closely approximated by a function linear in $\log(SAS(N))$. Figure 4.1.3-3 plots $cpu(N)$ vs. $\log(SAS(N))$, using the data in the above table. Although little can be concluded with confidence from this plot, the data do not support the hypothesis that cpu -time grows linearly with the log of the size of the assignment space.

4.2 New General Algorithms Combining Backtracking and Constraint Satisfaction

4.2.1 BACKMARK: Backtrack With Fewer Redundant Pair-Tests

In this section we define a new general backtrack-type algorithm for SAPs that executes a subset of the pair-tests executed by BACKTRACK. At the price of a relatively modest amount of additional storage over that required by BACKTRACK, BACKMARK executes exactly the same distinct pair-tests executed by BACKTRACK, but recomputes each one fewer times or no more than the same number of times as does BACKTRACK. In this section we only define BACKMARK and motivate its discovery. Measurements comparing its performance to that of the other algorithms tested in this chapter are deferred until Section 4.3.

The plots of the redundancy ratio measure $M(N) = T(N) / D(N)$ for BACKTRACK (i.e., Figures 4.1.1-3 and 4.1.2-3) suggest that significant improvements in performance might be realized if redundant pair-tests could be eliminated somehow. Viewing this as a store vs. recompute issue suggests an attempt to devise a new algorithm whose time/space tradeoff differs from that of BACKTRACK.

Any pair-test that is recomputed is redundant, but the practical issue is how to achieve the effect of storing the pair-test results without requiring much additional storage. In seeking a good time/space tradeoff for a new algorithm, it is useful first to look at extreme cases. Certainly, the classical backtrack algorithm could be modified to store the result of each pair-test in a large table, so as to perform each pair-test at most once, but the memory requirements may be prohibitive for large problems, since as many as $D_{\max}$ values may need to be stored.¹⁶ For the 8 queens problem $D_{\max} = 1568$; for 12 queens, $D_{\max} = 8712$. For a moderately large problem of, say, 20 variables with 50 values each, $D_{\max} = 475000$. In contrast, the VALUES array of BACKTRACK contains for these three specific examples contains $\sum_{1 \leq i \leq N} k_i = 60, 138, \text{ and } 1000$ elements, respectively.

Besides requiring a non-negligible amount of additional space, this hypothetical "large table" version of the backtrack algorithm may not significantly reduce the total amount of

¹⁶ For example, compare $D_a(N)$ or $D_f(N)$ with $D_{\max}(N)$ in Figure 6.1.1-2.

cpu-time required, since only the problem dependent term cpu-pd is reduced, and not the value of the problem independent term cpu-pi . That is, such a scheme reduces the average cost per pair-test execution, but not the number of pair-tests executed. Here we seek instead an algorithm having a favorable time/space tradeoff, and one that eliminates invocations of some pair-tests that BKRAK executes.

Consider the trace of BACKTRACK for 8-queens given in Section 4.1.1. In the portion marked "A", for example, value (6,1) of problem variable (p.v.) 6 (indicating queen 6 is placed in row 6, column 1) is tested against the current instantiated candidate value of p.v. 1, namely (1,1). Since this pair-test fails, we proceed to the next candidate value of p.v. 6, namely (6,2), which succeeds against (1,1) and (2,3) but not against (3,5). In no case in portion "A" do all five pair-tests succeed, so backtrack occurs, followed subsequently by the successful instantiation of value (5,8) of variable 5, followed by portion "B" in the trace.

The fact that 19 pair-tests are performed in both portion "A" and in portion "B" is no coincidence--it is guaranteed. During portion "B" not one new pair-test is executed; the results were computed during portion A. This follows from the fact that the backtracking that occurred between portions A and B changed only the assignment of p.v. 5, leaving the assignments of p.v.s 1 through 4 unchanged. Hence the only pair-tests in portion B that could be distinct from those of portion A are those involving p.v. 5, but none were executed in portion A, so none can be executed in portion B either. Similarly, in portion D all but the rightmost pair-test of (5,2) are redundant, having been executed during portion C. (The "rightmost" pair-test of (5,2) in portion D is the pair-test (5,2,4,7).) Thus by the end of portion D what has cost 90 pair-tests could have cost only $90 - 19 - 4 = 67$, if the results of the other pair-tests had not been recomputed.

Some of the pair-tests executed redundantly in portion B are recomputed yet again in portion E of the same trace. Portions A differs from portion B in the assigned location of queen 5, but those of queens 1, 2, 3, and 4 are the same. Portions A and B differ from portion E in the assigned locations of queens 4 and 5, but not in those of queens 1, 2, and 3. After portion E, 101 pair-tests have been executed, only 69 of which are distinct.

Algorithm BACKMARK, defined below by a SAIL procedure, eliminates in a general way all of the redundant pair-tests pointed out above. The comments concerning the choice of actual parameters for invocations of algorithm BACKTRACK that precede its code in Section

4.1.1 also apply to BACKMARK, subject to the following qualifications. Top level invocation for 8-queens takes the form

$tmp \leftarrow \text{BACKMARK}(1, 8, A, B, C, D),$

with array dimensions $D[1:N]$ and $C[1:N, 1:kmax]$. The initial value of each element of arrays C and D is required to be 1. Underlining below indicates the additions to BACKTRACK required to obtain BACKMARK, except that "for $i \leftarrow \text{new}[var] \dots$ " in BACKMARK is replaced by "for $i \leftarrow 1 \dots$ " in BACKTRACK.

```

recursive integer procedure backmark(integer var, n;
                                integer array a, k, mark, new);
! n = no. of problem variables;
! k[i] = no. of candidate values of problem variable i;
! var is the problem variable instantiated during current invocation;
! mark and new contain information to eliminate recomputations;
begin
  integer i, val;
  boolean testflg;
  for val  $\leftarrow 1$  step 1 until k[var] do
    if mark[var, val] geq new[var] then
                                     ! if this val has any chance at all, then...;
      begin "place-and-test"
        testflg  $\leftarrow$  true;
        for i  $\leftarrow$  new[var] step 1 while i < var and testflg do
          testflg  $\leftarrow$  pairtest(i, a[i], var, val);
                                     ! Does new queen on this square attack queen i on its assigned square?;
          mark[var, val]  $\leftarrow$  i - 1;           ! number of successful tests;
        if testflg then           ! if passed all tests, then...;
          begin "instantiate"
            a[var]  $\leftarrow$  val;
            if var = n then return(-1)           ! found solution, so unwind;
            else if backmark(var+1, n, a, k, mark, new) = -1 then return(-1)
                                     ! if son says to unwind, then tell father to unwind;
          end "instantiate"
        end "place-and-test";
                                     ! this var can't be instantiated;
      new[var]  $\leftarrow$  var - 1;           ! reset state of this var...;
      for i  $\leftarrow$  var + 1 step 1 until n do           ! ...and others;
        if new[i] > new[var] then new[i]  $\leftarrow$  new[var];
      return(0)           ! backtrack and continue search;
    end;
  end;

```

Note that the only difference in control sequencing between BACKTRACK and BACKMARK is the addition of "if mark[var, val] geq new[var] then" as a condition for executing the block labeled "place-and-test". The value $\text{mark}[i, j] = k$ indicates that the last

time candidate value v_{ij} was "pair-tested" against the instantiated values of the problem variables having indices less than i , the final pair-test to be executed was against the instantiated value of problem variable k , i.e., the pair-test $(k, A[k], i, j)$. (Furthermore, we know that that pairtest invocation returned false if $k < i-1$, or returned either true or false if $k = i-1$.) Note also that there are two conditions under which pair-tests executed by BACKTRACK are avoided by BACKMARK: (a) if the condition preceding the block labeled "place-and-test" is not true and (b) if $\text{new}[\text{var}] > 1$ when the iteration containing the invocation of procedure pairtest is executed.

The behavior of BACKMARK can be illustrated using the same 8-queens partial trace previously referred to, which is reproduced below with annotations indicating the values of the arrays mark and new.

The annotated trace is interpreted as follows, using the line beginning "3,2" as an example:

- 1) The parenthesized expression "(1:1-y)" indicates that $\text{mark}[3,2] = 1$ (number to left of colon), and that $\text{new}[3] = 1$ (number to right of colon).
- 2) The symbol to the right of the dash indicates whether block "place-and-test" was executed ("Y" = yes, "N" = no).
- 3) The number in brackets is the value assigned to $\text{mark}[3,2]$ by the assignment statement after the iteration of pair-test invocations. (Compare this number with the number preceding the colon.)
- 4) Farther on in the trace, the symbol "a" indicates a pair-test executed by BACKTRACK but not by BACKMARK due to condition a mentioned above, and similarly the symbol "b" indicates a pair-test avoided under condition b.
- 5) All assignments to elements of array new are also shown in the trace in correct chronological sequence.

Incomplete trace of algorithm BACKMARK for 8-Queens SAP:
 x,y = queen x on square at row x, column y

annotations explained in immediately preceding text.

1,1

```

2,1 (1:1-Y) F [1]
2,2 (1:1-Y) F [1]
2,3 (1:1-Y) T [1]
3,1 (1:1-Y) F [1]
3,2 (1:1-Y) TF [2]
3,3 (1:1-Y) F [1]
3,4 (1:1-Y) TF [2]
3,5 (1:1-Y) TT [2]
4,1 (1:1-Y) F [1]
4,2 (1:1-Y) TTT [3]
5,1 (1:1-Y) F [1]
5,2 (1:1-Y) TTTF [4]
5,3 (1:1-Y) TF [2]
5,4 (1:1-Y) TTTT [4]
6,1 (1:1-Y) F [1]
6,2 (1:1-Y) TTF [3]
6,3 (1:1-Y) TF [2]
6,4 (1:1-Y) TTTF [4]
6,5 (1:1-Y) TTF [3]
6,6 (1:1-Y) F [1]
6,7 (1:1-Y) TF [2]
6,8 (1:1-Y) TTF [3]

```

new[6] ← 5

```

5,5 (1:1-Y) F [1]
5,6 (1:1-Y) TF [2]
5,7 (1:1-Y) TTF [3]
5,8 (1:1-Y) TTTT [4]
6,1 (1:5-N) a
6,2 (3:5-N) aaa
6,3 (2:5-N) aa
6,4 (4:5-N) aaaa
6,5 (3:5-N) aaa
6,6 (1:5-N) a
6,7 (2:5-N) aa
6,8 (3:5-N) aaa

```

new[6] ← 5

new[5] ← new[6] ← 4

```

4,3 (1:1-Y) TF [2]
4,4 (1:1-Y) F [1]
4,5 (1:1-Y) TF [2]
4,6 (1:1-Y) TTF [3]
4,7 (1:1-Y) TTT [3]
5,1 (1:4-N) a
5,2 (4:4-Y) bbbT [4]
6,1 (1:4-N) a
6,2 (3:4-N) aaa
6,3 (2:4-N) aa
6,4 (4:4-Y) bbbT [5]

```

7,1	(1:1-Y)	F	[1]
7,2	(1:1-Y)	TTTTF	[5]
		.	
		.	
		.	

We claim without formal proof that BACKMARK is functionally equivalent to BACKTRACK in the sense that for any SAP the solution assignments (if any exist) found by the two algorithms are identical. We further conjecture for any SAP that BACKMARK executes exactly the same distinct pair-tests as BACKTRACK (so that $D_{\text{BACKMARK}} = D_{\text{BACKTRACK}}$), and that each such distinct pair-test is executed at least as many times by BACKTRACK as by BACKMARK (i.e., $T_{\text{BACKMARK}} \leq T_{\text{BACKTRACK}}$). (These two conditions imply that $M_{\text{BACKMARK}} \leq M_{\text{BACKTRACK}}$.) Proving these conjectures remains a matter for future work. Sections 4.3 and 4.4 compare the performance of BACKMARK with those of other algorithms.

4.2.2 BACKJUMP: Backtrack that Jumps Multiple Levels

We define now a new general backtrack-type algorithm for SAPs, here called BACKJUMP, that sometimes backtracks across multiple levels of the search tree instead of across only a single level. BACKJUMP is the result of an attempt to produce the domain-element-elimination effect of DEE-0 in a backtrack-like control context. DEE-0 eliminates candidate values when it detects a global inconsistency; BACKJUMP does so in the context of candidate values already instantiated higher in the search tree.

Specifically, as noted in Section 4.1.1, in DEE-0 a candidate value $z \in R_i$ of problem variable x_i is eliminated when it is detected for some other problem variable x_j that $\nexists y \in R_j$ such that $(z,y) \in P_{ij}$ (condition 4.1.1-1). In BACKTRACK, however, the truth value of condition (4.1.1-1) can go undetected. This will occur for example if any of the pair-tests between any of candidate values $y_1, y_2, \dots, y_{k_j}$ on the one hand and any of the instantiated candidate values $A(1), A(2), \dots, A(i-1)$ on the other hand, fails. In such an event, involving say y_3 , the pair-test (j, y_3, i, z) is not executed and hence the truth value of (4.1.1-1) remains unknown.

We observe that detection of a condition different from and weaker than (4.1.1-1) admits a similar effect. If each of the candidate values $y_1, y_2, \dots, y_{k_j}$ fails against either $A(1), A(2), \dots, A(i-1)$, or $A(i)$, then no assignment having the latter candidate values as its first i

elements can be a solution. Hence there is no reason not to backtrack immediately to level i under such circumstances.

The incomplete trace of BACKTRACK given in Section 4.1.1 provides a concrete example of the above effect. During the invocation at level 6 (i.e., portion "A" in the trace), each of the k_6 (i.e., 8) values of x_6 is tested in turn against the instantiated candidate value of problem variables x_1, x_2, x_3, x_4 , and x_5 . In the case shown, however, none of these values satisfies each of P_{16}, P_{26}, P_{36} , and P_{46} , so a backtrack occurs and search proceeds in the portions "C2" and "B". However, it is necessarily the case that no solution can be found in portions "C2" and "B", since the instantiated candidate values of x_1, x_2, x_3 , and x_4 have not changed. Hence the pair-tests performed in these portions are unnecessary and may be eliminated in favor of proceeding directly with search in portion "F". Hence an algorithm that does not backtrack always not to the preceding level, but sometimes jumps across several levels, to level 4 in this example, might be more efficient than BACKTRACK. Such an algorithm needs to determine which level to jump back to.

The following SAIL procedure instantiates such an algorithm, here called BACKJUMP, in a form that halts after finding a first solution. Procedure BACKTRACK is defined the same, minus the underlined portions, except that the statement "return(returndepth)" in BACKJUMP is replaced by "return(0)" in BACKTRACK, and except for minor differences in the program block which contains the recursive call to BACKJUMP.

```

recursive integer procedure backjump(integer var, n; integer array a, k);
begin
integer i, val, returndepth, faildepth;
boolean testflg;
returndepth  $\leftarrow$  0;
for val  $\leftarrow$  1 thru k[var] do           ! check each candidate value of the var'th problem variable;
begin
testflg  $\leftarrow$  true;
for i  $\leftarrow$  1 step 1 while i  $\leq$  var and testflg do      ! test this c.v. against each instantiated c.v.;
testflg  $\leftarrow$  pairtest(i, a[i], var, val);
if not testflg then faildepth  $\leftarrow$  i - 1;           ! note: uses final value of loop variable i;
if testflg then           ! if passed all tests, then ...;
begin
a[var]  $\leftarrow$  val;
if var = n then return(-1)           ! solution found, so unwind to outermost call;
else
begin
faildepth  $\leftarrow$  backjump(var+1, n, a, k);
if faildepth  $\leq$  var then return(faildepth) ! unwind to level given by value of faildepth;
end
end;
returndepth  $\leftarrow$  returndepth max faildepth
end;
return(returndepth)           ! backtrack and continue search;
end;

```

Using the trace of algorithm BACKTRACK given in Section 4.1.1 to illustrate the behavior of BACKJUMP, consider the first invocation of BACKJUMP at level 6 (i.e. portion A in the trace). As each of the 8 candidate values of problem variable x_6 are considered in turn, the instance of program variable returndepth local to that invocation takes on the successive values 1,3,3,4,4,4,4,4. Hence this invocation returns the value 4, which is assigned to the instance of program variable faildepth local to the parent invocation at level 5. At this point the condition "faildepth $\leq$ var" is satisfied (i.e., $4 \leq 5$), and this invocation immediately returns the value 4, eliminating the pairtests shown in portions C2 and B in the trace. Among the subsequent pair-tests shown in that partial trace, BACKJUMP eliminates none.

We claim without formal proof that this algorithm, which we call BACKJUMP, is functionally equivalent to BACKTRACK and to BACKMARK in the sense that for any SAP the solution assignments found by the three algorithms (if any exist) are identical. We further claim for any SAP that every distinct pair-test executed by BACKJUMP is also executed by BACKTRACK (i.e., $D_{\text{BACKJUMP}} \leq D_{\text{BACKTRACK}}$), and that each such pair-test is executed at least as many times by BACKTRACK as by BACKJUMP ($T_{\text{BACKJUMP}} \leq T_{\text{BACKTRACK}}$).

We attempt no formal analysis of BACKJUMP in this dissertation, observing informally only that the number of pair-tests eliminated by BACKJUMP, as compared with BACKTRACK, depends in some way on the number of occurrences of multiple level jumps and on the number of levels jumped over, which is clearly bounded above by N . Sections 4.3 and 4.4 compare the performance of BACKJUMP with those of other algorithms.

4.2.3 DEELEV(i): Constraint Satisfaction After Backtracking to Level i

As noted in Section 4.1.1, DEEB operates by first applying DEE-0 to the given SAP S_0 , which results in a new SAP S_0' in which each problem variable x_i has k_i' candidate values, where $0 \leq k_i' \leq k_i$. To solve S_0' , DEEB instantiates problem variable x_1 to one of its k_1' candidate values, whereby S_0' becomes the new more constrained SAP S_1 . DEE-0 is applied to the latter, producing as a result the SAP S_1' , and backtracking continues in like manner. One might speculate on intuitive grounds that the effect of introducing additional constraint by instantiation is to cause DEE-0 to terminate more quickly, and that the efficiency of the various invocations of DEE-0 increases with depth in the backtrack tree.

Figure 4.2.3-1 presents relevant evidence concerning this hypothesized efficiency by plotting as a line segment the "before and after" SAS and cumulative T values of each DEE-0 invocation during the search of the 8-queens SAP using "left-to-right" c.v. ordering. Using the superscript and subscript notation above, the line segments in the figure connect the points (SAS_0, T_0) with (SAS_0', T_0') ; (SAS_1, T_1) with (SAS_1', T_1') , and so on, where $T_0 = 0$, $T_i = T_{i+1}'$, $SAS_{i+1} = SAS_i' / k_i'$, and $T_i' - T_i$ is the number of pair-tests executed during an invocation of DEE-0 at level i of the DEEB backtrack search tree. (This notation does not distinguish the various invocations of DEE-0 occurring at levels 2 and 3 in this example, but the meaning of Figure 4.2.3-1 is presumably clear.) So the slope of each line segment in this figure measures the efficiency of the corresponding DEE-0 invocation: the slope measures the rate that $\log(SAS)$ is reduced per pair-test executed. Formally, the slope has value given by the formula $\log(SAS/SAS') / (T' - T)$, where for simplicity the subscripts are omitted. By this measure, the figure indicates that efficiency of DEE-0 does indeed increase with depth in the tree.

Figure 4.2.3-1 suggests that DEEB be modified by eliminating the invocation of DEE-0 that precedes the instantiation of problem variable x_1 by DEEB. Here we generalize this

notion by defining a parameterized algorithm, here called DEELEV(*i*), that executes BACKTRACK until a consistent partial assignment A_i of the first *i* problem variables is found, whereupon DEEB is invoked with this partial assignment as input. SAIL procedure DEELEV is defined the same as BACKTRACK, except that

BACKTRACK(integer var, n; integer array a, k)

in the procedure head is replaced by

deelev(integer var, n, lev; integer array a, k)

and the statement

if var = n then return(-1)

else if BACKTRACK(var+1, n, a, k) = -1 then return(1)

is replaced by the statement

if var = lev then

begin if deeb(lev+1, n, a, k) = -1 then return(-1)

end

else if deelev(var+1, n, lev, a, k) = -1 then return(-1)

Note that DEELEV can be defined alternatively using BACKMARK or BACKJUMP in place of BACKTRACK, but the results given below assume the definition given above.

Figure 4.2.3-2 compares DEEB with DEELEV(1) (i.e., DEELEV invoked with actual parameter LEV = 1) and DEELEV(2) by mean $T_f(N)$ for N-Queens problems. The sample set of problem instances in this case is the same set of randomly generated candidate value orderings described in Section 4.1.2 (30 to 100 samples per value of N). The ratio of $T_f(N)$ using DEEB to $T_f(N)$ using DEELEV(1) ranges from 1.29 for N = 16 to 2.42 for N = 5. Excluding the case N = 4, the ratio of $T_f(N)$ using DEELEV(1) to $T_f(N)$ using DEELEV(2) ranges from 1.29 for N = 15 to 2.06 for N = 5. Figures 4.2.3-3 and 4.2.3-4 show co-responding values of mean $D_f(N)$ and mean $M_f(N)$, respectively. These data indicate a reduction in $T_f(N)$ for DEELEV(1) and DEELEV(2) over that of DEEB. They also show that this reduction reflects reductions in both $D_f(N)$ and in $M_f(N) = T_f(N) / D_f(N)$.

Figures 4.2.3-5 and 4.2.3-6 show how performance of DEELEV(*i*) varies with *i*, for fixed

N , namely for $N = 10$ and also for $N = 12$. Note that the data given for $i = 0$ are those obtained using DEEB. The data indicate that mean $T_f(N)$ decreases monotonically with i for $i < 5$ for both $N = 10$ and $N = 12$. For $i \geq 5$, $T_f(N)$ increases monotonically with i in the case $N = 12$, and increases and then decreases with i in the case $N = 10$. For both $N = 10$ and $N = 12$, $D_f(N)$ decreases with i over the entire range of i . These data indicate that $T_f(N)$ does vary with i , for fixed N , but leave uncertain how to predict a priori for a given SAP the value of i that minimizes $T_f(N)$.

One possible heuristic is to choose $i = N/2$. Figure 4.2.3-7 plots the observed values of mean $T_f(N)$ for $N = 4, 6, 8, \dots, 18$ executing DEELEV($N/2$). The sample of N -queens instances tested here is the same as that in Sections 4.1.2 and 4.3, namely the random candidate value orderings. Also plotted in Figure 4.2.3-7 for purposes of comparison are $D_f(N)$ observed under the same conditions and $T_f(N)$ using DEEB. These data suggest the use of $N/2$ as a default value for LEV, if no other available information about the problem to be solved suggests a different value.

4.3 Comparative Performance Measurements for N-Queens SAPs

Having defined four algorithms for SAPs, we now compare their respective performances under identical conditions for a large sample set of problems. Figure 4.3-1 compares the mean number of pair-tests (i.e., mean $T_f(N)$) executed by algorithms BACKTRACK, BACKMARK, BACKJUMP, and DEEB, respectively, to find a first solution for N-Queens SAPs.¹⁷ The sample set of SAPs over which these measurements are taken is the same for each algorithm, namely the set described in Section 4.1.2 of $m(N)$ randomly selected candidate value orderings for $N = 4, 5, \dots, 17$.

We observe in Figure 4.3-1 that among the four algorithms being compared, BACKMARK executes the fewest pair-tests on the average, followed in order by BACKJUMP, BACKTRACK, and DEEB, and that this ordering among the four algorithms is observed to hold for each value of N . Note also that the performance of BACKJUMP differs very little from that of BACKTRACK. Note also that $T_f(N)$ for BACKMARK is much less than for the other three algorithms, but also much greater than $T_{\min}(N)$. Note also that none of the four curves is closely approximated by a straight line in this semilog plot, as would be the case if $T_f(N)$ grew exponentially as Mackworth [1977, p. 100] suggests. Note also the relatively poor performance of algorithm DEEB: we have identified one set of SAPs for which DEEB is less efficient than BACKTRACK by the mean $T_f(N)$ measure for each value of N observed, and much less efficient than BACKMARK under identical conditions. These data do not support Mackworth's suggestion that Waltz-type algorithms are "clearly more effective than automatic backtracking" [Mackworth 1977, p. 116]. The factors to which this inefficiency can be attributed remain uncertain at present (but see Sections 4.2.3 and 4.4 for additional data).

Now we can also determine whether the difference in performance between "left-to-right" candidate value ordering and random c.v. ordering, noted in Figures 4.1.2-2 and 4.1.2-3 is peculiar to BACKTRACK, or is characteristic of BACKMARK, BACKJUMP, and DEEB as well. Figure 4.3-2 plots the ratios of the value of $T_{f|ALG}(N)$ using "left-to-right" candidate value ordering to the corresponding observed value of mean $T_{f|ALG}(N)$ using random c.v. ordering, where ALG denotes either BACKTRACK, BACKMARK, BACKJUMP, or DEEB. The ratio values

¹⁷ The values constituting the curves labeled "BACKTRACK" and "BACKMARK" in this figure are taken from Figure 1 in Gaschnig [1977b]. The curve labelled "BACKTRACK" is identical to the one labelled " $T_f(N)$ (mean)" in Figure 4.1.2-1.

plotted in Figure 4.3-2 indicate apparently that differences in performance between random c.v. ordering and "left-to-right" c.v. ordering are exhibited by each of the four algorithms, and that these differences are generally much larger for $14 \leq N \leq 16$ than for $N < 14$. For the case of BACKMARK at $N = 20$, the ratio is $1343970 / 2696.7 = 498.3!$

We now compare the four algorithms by $D_f(N)$ rather than by $T_f(N)$. Figure 4.3-3 plots the corresponding mean values of $D_f(N)$ for the same experiments whose results are shown in Figure 4.3-1. No curve is plotted for BACKJUMP in Figure 4.3-3; the mean value of $D_f(N)$ using BACKJUMP is observed to differ from mean $D_f(N)$ using BACKTRACK by no more than 5% over $N = 4, 5, \dots, 15$. Figure 4.3-4 plots the corresponding mean values of the redundancy ratio $M_f(N) = T_f(N) / D_f(N)$ collected during the same experiments.

To provide further evidence for larger values of N , we extended the previous experiment to the cases $N = 20, 25, 30, 35, 40$, and 50 with $m(N) = 50$ samples per value of N selected in the same manner as above, and we measured mean $T_f(N)$ for BACKMARK only. Combining these data with some of the those for BACKMARK in Figure 4.3-1, we observed the mean $T_f(N)$ values tabulated below.

N	mean $T_f(N)$	$T_{\min}$	$D_{\max}$	SAS
5	23.6	10	210	1875
10	542	45	4050	5×10^9
15	1513	105	22155	2×10^{17}
20	2696	190	72200	5×10^{25}
25	4715	300	180300	5×10^{34}
30	11520	435	378450	1×10^{44}
35	28415	595	708645	6×10^{53}
40	21890	780	1216800	6×10^{63}
50	55020	1225	3001250	4×10^{84}

Approximating the above $T_f(N)$ values by the formula $T_f(N) = N^{C(N)}$ and solving for $C(N)$, we obtain the formula $C(N) = \log T_f(N) / \log N$. For the above list of values of N and mean $T_f(N)$, the values of $C(N)$ are 1.964, 2.734, 2.704, 2.637, 2.628, 2.750, 2.709, and 2.79, respectively. These are plotted in Figure 4.3-5. Note that with the exception of $N = 5$, these $C(N)$ values fall in the interval 2.75 ± 0.14 . Note that our purpose in presenting this approximation is simply pragmatic: to show how well a particular approximation fits the observed data, without suggesting that the approximation is valid generally. Pragmatically, the data and approximation would seem to cast doubt on the proposition that mean $T_f(N)$ for BKMARK grows exponentially with N in this case.

All life is an experiment. The more experiments you do the better.

Ralph Waldo Emerson

4.4 Experimental Results for Randomly Generated SAPs

4.4.1 SAP Equivalence Classes Parameterized by Size and by "Degree of Constraint" (L)

Thus far we have compared the performance of five algorithms, by each of three performance measures, for several sample sets of N-queens problems. Now we wish to determine the extent to which these results generalize to SAPs other than the N-Queens SAPs. Section 4.1.1 enumerates a variety of particular SAPs that could be subjected to experiment. However, the case study approach of course is limited in that a large number of individual case studies may be required to reveal credible generalizations.

For this reason, in this section we generalize our experiments beyond N-Queens SAPs to a class of randomly generated SAPs. This approach also has limitations, in that the performance of a given algorithm in solving randomly generated problems may be quite different from that in solving a problem representing some particular situation arising in practice. This question has obvious importance for analysis of algorithms research: a formula derived for an algorithm's average performance, say, over an aggregate of random samples may not be useful in predicting its performance on cases arising in practice.

Hence it is important not only to measure the performance of algorithms for randomly generated problems, but also to determine how the characteristics of random problems differ from those of "particular-structure" problems. Accordingly, we attempt to provide evidence concerning both these issues. Our approach is to define a parameterized equivalence relation on the set of all possible SAPs, partitioning this set into (disjoint) equivalence classes so that any particular problem, the 8-Queens SAP, say, belongs to some particular equivalence class. Then we shall determine how typical the 8-queens SAP is with respect to the other member of the equivalence class to which it belongs.

The partition on the set of all SAPs used here is such that members of a given equivalence class have identical values of the parameters representing size and "degree of constraint" of a SAP. We then define a procedure for generating a sample set of SAPs selected randomly (independently, uniformly, and with replacement) from among the members of a specified equivalence class. We use this procedure to generate randomly for each N a set of SAPs each of whose size and degree of constraint parameters matches that of the N -Queens SAP to which it corresponds (one parameter set per value of N).

Definition 4.4-1.

- a) SAPs $S = \{N, R_1, R_2, \dots, R_N, P_{12}, P_{13}, \dots, P_{N-1, N}\}$ and $S' = \{N', R'_1, R'_2, \dots, R'_{N'}, P'_{12}, P'_{13}, \dots, P'_{N'-1, N'}\}$ are N -similar if and only if $N = N'$.
- b) SAPs S and S' are N - k_i -similar if and only if S and S' are N -similar and $k_i = k'_i$ (where $k_i = |R_i|$ and $k'_i = |R'_i|$), for each $i = 1, 2, \dots, N$.

For example, let the 8-Queens-Knights SAP be defined like the 8-Queens SAP except that the "chess pieces" in the former move either as queens or as knights. Then the 8-Queens SAP and the 8-Queens-Knights SAP are N - k_i -similar.

We define the "degree of constraint" of a SAP to be the fraction of distinct pair-tests for that SAP that have the value "true". Formally, given a SAP S ,

$$L_S = \frac{\sum_{1 \leq i < j \leq N} \sum_{i < j \leq N} |P_{ij}|}{\sum_{1 \leq i < j \leq N} \sum_{i < j \leq N} k_i \cdot k_j}$$

So $0 \leq L \leq 1$ by definition.

Definition 4.4-2

SAPs S and S' as in Definition 4.4-1 are N - k_i - L -similar if and only if S and S' are N - k_i -similar and $L_S = L_{S'}$.

We use the following procedure to select randomly (independently, uniformly, and with replacement) a SAP having specified values of N , $k_1, k_2, \dots, k_N$, and L . For each i and j such that $1 \leq i < j \leq N$, we create a boolean-valued matrix M_{ij} of size $k_i \times k_j$. To each of the $D_{\max}$ elements of each such matrix we assign (by means of pseudo-random number generator) the value "true" with probability L , and the value "false" with probability $1 - L$.¹⁸

4.4.2 N-Queen's SAPs vs. "Random-N-Queens" SAPs: Comparative Algorithm Performance

Using the technique described in Section 4.4.1, now we generate a sample set of "random-N-Queens" SAPs corresponding parametrically with the N-Queens SAPs. By exhaustively enumerating the set of $D_{\max}$ distinct pair-tests and counting the number of those that have the value "true", the values of L for the N-queens problems for $N = 4, 5, \dots, 16$ are determined to be (to 3 decimal places) 0.444, 0.552, 0.622, 0.676, 0.714, 0.746, 0.770, 0.791, 0.808, 0.823, 0.835, 0.846, 0.856, respectively. Figure 4.4.2-1 plots these values against N , showing that L appears to grow with N as a smooth curve. The sample set of what we shall call "random-N-queens" SAPs consists of: 50 independently and randomly generated SAPs having $N = k_2 = k_3 = k_4 = 4$, $k_1 = 2$, and $L = .444$; a similar set of 50 SAPs is generated for each of $N = 5, 6, 7$ using the corresponding values of L enumerated above; 100 such samples each of $N = 8, 9, 10, 11, 12$; 150 samples for $N = 13$; and 250 samples for $N = 14$, for a total of 1100 problem instances in the sample set.

Figure 4.4.2-2 shows the mean values of $T_f(N)$ observed using algorithms BACKTRACK, BACKMARK, BACKJUMP, and DEEB to find first solution for the randomly generated SAPs in the random-N-queens sample set. Comparing these data with those in Figure 4.3-1, note that the relative ordering of the algorithms is the same in both figures: of the four tested algorithms BACKMARK executes the fewest pair-tests on the average for each value of N , followed in order by BACKJUMP, BACKTRACK, and DEEB.

To compare more easily the values shown in Figure 4.4.2-2 with the corresponding values in Figure 4.3-1, Figure 4.4.2-3 plots the ratio of each value plotted in the latter figure to its corresponding value in the former figure. The values so plotted represent the results of than 7340 distinct algorithm executions (3440 represented in Figure 4.3-1 and 3900 represented in Figure 4.4.2-2). We observe that the differences between $T_f(N)$ using DEEB and $T_f(N)$ using BACKTRACK are larger for random-N-queens SAPs than for the corresponding N-queens SAPs, and that the magnitude of this difference grows with N , and

¹⁸ Note that using this procedure the fraction L' of matrix elements assigned the value "true" does not necessarily equal exactly the given value of L , but rather approximates it. The law of large numbers insures that the difference between L and L' is negligible in the present cases.

that the same holds for the corresponding differences between the performance of BACKTRACK and that of BACKJUMP. Figure 4.4.2-3 shows that the N-queens sample set and the random-N-queens sample set are sharply more distinguishable for $N \geq 10$ than for $N < 10$ (i.e., the values plotted in this figure differ sharply from the value 1 in the former case), and are sharply less distinguishable by algorithm DEEB than by the other three algorithms. Note in particular that for $N \geq 10$, N-Queens SAPs require many more pair-tests to be executed on the average than is the case for the corresponding random-N-queens SAPs. (Why this should be the case remains unknown.)

Figure 4.4.2-4 compares the 10-Queens SAP with the "10-random-Queens" SAP by frequency distribution of T_f values using BACKMARK. The ratio of the means of the two distributions given in this figure is the value plotted in Figure 4.4.2-2 for BACKMARK at $N = 10$.

In this section the SAPs tested were selected randomly from specific $N-k_i-L$ equivalence classes, as defined in Section 4.4.1. Clearly one can specify additional parameters, such as the number of solutions, thereby further partitioning these classes. This approach makes possible a finer-grained investigation of problem structure. Several such possibilities are suggested under item E4-2 in Appendix B.

4.4.3 Cost as a Function of L: A Sharp Peak at $L \sim 0.6$

Next we report the results of experiments designed to show how the cost of solving a SAP depends on the degree of constraint possessed by the problem, all other things being equal. From only the results plotted in the figures of Sections 4.3 and 4.4.2, it is difficult to infer the dependence of the mean value of $T_f(N)$ on the value of L , because the SAPs in the N-queens sample set differ among each other both in size (i.e., N and k_i values) as well as in L values, and the same holds for the random-N-queens sample set. Moreover, L ranges only from 0.444 to 0.856 among the N-Queens SAPs or Random-N-Queens in these two sample sets (i.e., reflecting that $4 \leq N \leq 17$).

Accordingly, we performed experiments analogous to those whose results are plotted in Figures 4.3-1, 4.3-3, and 4.3-4, using a sample set of randomly generated SAPs that are identical to each other in size but differ systematically in value of L . Specifically, we used the method described in Section 4.4.1 to generate randomly 150 SAPs, each having $N = 10$,

$k_1 = k_2 = \dots = k_{10} = 10$ and link percentage value $L = 0.1$; and to generate analogously a set of 150 distinct SAPs for each of $L = 0.2, 0.3, \dots, 0.9$. For these values of N and the k_i , $T_{\min} = 45$, $D_{\max} = 4500$, and $SAS = 10^{10}$. For each of these 1350 SAPs, we measured T_f , D_f , and derived M_f .

The four curves plotted in Figure 4.4.3-1 show the mean values of $T_f(L)$ observed when each of the algorithms BACKTRACK, BACKMARK, BACKJUMP, and DEEB, respectively, is applied to each of the SAPs in this "Identical size, varying L " (ISVL) sample set. Figures 4.4.3-2 and 4.4.3-3 show the corresponding mean values of $D_f(L)$ and $M_f(L) = T_f(L) / D_f(L)$ observed for the ISVL sample set.

The T_f values plotted in Figure 4.4.3-1 for the boundary cases $L = 0.0$ and $L = 1.0$ are derived analytically rather than observed experimentally. The values plotted are $T_f = k_1 * k_2 = 100$ at $L = 0.0$ for all four algorithms, and $T_f = N(N-1) / 2 = 45$ at $L = 1.0$ for BACKTRACK, BACKMARK, and BACKJUMP, and $T_f = 1305$ at $L = 1.0$ for DEEB.

The data plotted in Figures 4.4.3-1, 4.4.3-2, and 4.4.3-3 indicate the same relative ordering of the four algorithms as observed in preceding sections. Furthermore, the results indicate that the number of steps executed (mean $T_f(L)$) depends strongly on degree of constraint (L), spanning a range whose extremes differ by a factor of 791 among the cases tested! For each of the four algorithms tested, a plot of the data suggests the existence of a single sharp peak in $T_f(L)$ at $L \sim 0.6$. The peak and range of performance of $T_f(L)$ are reflected in both $D_f(L)$ and $M_f(L)$ as well as in $T_f(L)$. These data show in particular that the Waltz-type algorithm does not better the others on the highly constrained problems tested.

Several extensions of this investigation of the dependence of cost on degree of constraint are proposed under Item E4-3 in Appendix B.

4.5 Other Results

4.5.1 Experimental Results for Map Coloring

The experimental results reported in the preceding sections of this chapter contradict Mackworth's predictions about the performance of Waltz-type algorithms, but these results may depend strongly on characteristics of the N-Queens and random-N-Queens problems. In particular, the experiments thus far have assumed SAPs having a complete constraint graph (i.e., a SAP for which all P_{ij} are proper). For contrast, in this section we compare the performances of algorithms BACKTRACK, BACKMARK, and BACKJUMP for SAPs that have an incomplete constraint graph, namely the problem of coloring with four colors the map depicted in Figure 4.5.1-1. (See Section 4.1.1 for a general description of map coloring SAPs, and for examples of other SAPs having incomplete consistency graphs.)

In these experiments, the 34-region map depicted in Figure 4.5.1-1 defines seven distinct SAPs: a SAP corresponding to the submap consisting of exactly the regions numbered 1 through 5; six other SAPs corresponding to the submaps consisting of exactly the regions numbered 1 through $N = 10, 15, 20, 25, 30,$ and $34,$ respectively. We define the colors $R_i = \{1,2,3,4\}$ for each SAP and each value of i .

For each of these seven SAPs we generate randomly 50 candidate value orderings in the manner described in Section 4.1.2 for N-Queens SAPs. Hence there are 50 algorithm executions (a.e.) per combination of algorithm and value of N ; hence 350 a.e. per algorithm, giving 1050 a.e. total. (The latter are not counted among the 13000 algorithm executions mentioned elsewhere in this chapter.) The mean $T_f(N)$ results are plotted in Figure 4.5.1-2. The values of BACKTRACK and BACKJUMP are observed to be identical except for the case $N = 34$. The large increase in $T_f(N)$ from $N = 30$ to $N = 34$ presumably reflects the fact that region 34 touches seven other regions, whereas in each of the six submaps the largest numbered region touches only three other regions. For BACKTRACK and BACKJUMP, we observed mean $M_f(N) = 1.0$ for $N = 5$; mean $M_f(N) < 1.1$ for $N = 10, 15, 20, 25,$ and 30 ; and mean $M_f(N) = 1.5$ for $N = 34$. For BACKMARK, we observed mean $M_f(N) < 1.001$ for each of the seven values of N tested.

These data are not voluminous enough to support very extensive generalizations, but do at least provide additional evidence that BACKMARK recomputes relatively few pair-tests.

4.5.2 Measures of Uniformity of Distribution of Solutions

To make performance more predictable it may be useful to attempt to discover more about the characteristics of the problem itself, independent of the choice of algorithm used to solve it. One characteristic of SAPs about which we are currently rather ignorant is how the solutions of a given SAP are distributed among the leaf nodes of a search tree for that SAP. More precisely, we seek here to determine how uniformly the solutions occur among a linear ordering of the set of assignments of the SAP, where the linear ordering reflects a particular candidate ordering.

This objective has practical import: Section 4.1.2 demonstrates (e.g., in Figure 4.1.2-1) that the efficiency of BACKTRACK can vary greatly with the particular candidate value ordering chosen, suggesting attempts to devise heuristics for choosing a candidate value ordering that minimizes the number of pair-tests required to find a solution. In attempting to devise such heuristics, it would seem relevant to know whether solutions tend to be distributed uniformly, as opposed to occurring in clumps. Investigations of this sort may also help account for the observed difference in performances between random candidate value ordering and some "left-to-right" candidate value ordering, such as the one considered in this chapter for the N-queens SAPs.

Definition 4.5.2-1

- a) Given a SAP $S = (N, R_1, R_2, \dots, R_N, P_{12}, P_{13}, \dots, P_{N,N-1})$, let π_i denote a permutation of the elements of R_i .
- b) Let π denote a tuple $(\pi_1, \pi_2, \dots, \pi_N)$.
- c) Then π determines a linear ordering $<_{\pi}$ of U_S , the set of assignments of S , such that $A_1 = (v_1, v_2, \dots, v_N) <_{\pi} A_2 = (y_1, y_2, \dots, y_N)$ if and only if the string $v_1 v_2 \dots v_N$ precedes the string $y_1 y_2 \dots y_N$ in the lexicographical ordering specified by π .
- d) Let $AR_{S,\pi}(A)$ denote the position of assignment A for SAP S in the ordering determined by π .

Since any subset of U_S is likewise linearly ordered by $<_{\pi}$, then in particular the subset of assignments that are solutions is linearly ordered by $<_{\pi}$.

Definition 4.5.2-2

- a) Let $s(i)$ denote the assignment corresponding to the i 'th solution under an ordering.
- b) Let the function $SD_{S,\pi}(i) = AR_{S,\pi}(s(i))$ be called the solution distribution function of S under π .

Figure 4.5.2-1 plots, in step function form, the values of i against $SD_{S,\pi_{\text{obv}}}(i)$ for the 5-queens SAP, where π_{obv} denotes the "left-to-right" candidate value ordering. The dashed line in this figure indicates the values that would be observed if solutions were distributed perfectly uniformly among assignments. Figures 4.5.2-2, 4.5.2-3, and 4.5.2-4 plot the analogous values for 7-Queens, 8-Queens and 9-Queens respectively.

The data in these figures indicate that the first solution of the 5-queens, 7-queens, and 8-queens SAPs occurs much later in the ordering of assignments imposed by our "left-to-right" candidate value ordering than would be the case if the solutions were distributed uniformly among the assignments. Stronger conclusions than this must await the results of future experiments, such as those proposed under item E4-6 in Appendix B.

4.5.3 Proof that $T_{\min}(N) = N(N-1)/2$

Here we describe our computational model in greater detail, to establish that if an algorithm B is valid for all SAPs, then B must execute at least $T_{\min}(N) = N(N-1)/2$ pair-tests for any SAP S having N problem variables and such that S has a solution. We achieve this by a simple adversary argument of the type described in Weide [1977, pp. 296-297] and in the references he cites.

The approach used here is to define an algorithm for SAPs abstractly as any function whose value, for a given SAP S satisfying Definition 4-1, is a tuple consisting of a finite sequence S of pair-tests, a boolean value C, and an assignment A for S in the case that C is "true". Our notion is that the algorithm executes the specified pair-tests, then terminates, claiming (by the value of C) either that no solution exists or that assignment A is a solution for S.

We assume that the pair-tests are executed in "black box" fashion: each pair-test in S is identified by a tuple (i,x,j,y) as in Definition 4-1, and from the black box or oracle is obtained a boolean value identifying whether $(x,y) \in P_{ij}$, e.g., in the case of N-Queens SAPs, whether two queens on specified squares attack each other. We assume that B is given as

input only the values of N and the k_i for the given SAP S and access to the oracle who answers pair-tests.

In this way we have identified the set of all algorithms for SAPs as a particular set of mathematical functions. Note that there can be in principle many implementations of a given algorithm for SAPs; in this computational model, however, we distinguish algorithms for SAPs solely on the basis of the values (S, C, A) they compute.

We term any such algorithm valid for SAPs if and only if for any SAP $S \in \langle \psi \rangle_{\text{SAP}}$, the claims C and A (if C is "true") are correct for S . To be valid, such an algorithm must give correct (i.e., "adversary-proof") answers for all SAPs, both all SAPs having one or more solutions and all SAPs having no solution. Below we establish a lower bound on the number of pair-tests required by any algorithm valid for SAPs in solving an arbitrary SAP, but this bound applies only to the set of all SAPs having a solution.

In the case that a given SAP S has a solution, we claim that any valid algorithm, B , for SAPs must necessarily produce a pair-test sequence S that contains the elements corresponding to the $N(N-1)/2$ pair-tests involving only the candidate values in the assignment A that algorithm B returns, i.e., the pair-tests $(1, A(1), 2, A(2)), (1, A(1), 3, A(3)), \dots, (N-1, A(N-1), N, A(N))$. Otherwise if one such assignment pair-test, call it (i_0, y_0, j_0, z_0) , is not contained in the pair-test sequence S , then an adversary can delete (y_0, z_0) from $P_{i_0 j_0}$, thus defining a different SAP S' for which algorithm B produces the same S and returns the same assignment A as algorithm B does for SAP S . However A is not a solution for S' , hence B is not valid, contradicting the assumption that B is valid. Note that this argument rests on the "black box" assumption: B decides which pair-test to execute next solely on the basis of the results it got executing the preceding pair-tests. B is not informed about the manner in which the truth value of an arbitrary pair-test is determined.

The $N(N-1)/2$ lower bound does not apply if the given SAP S has no solution. In this case, a valid algorithm for SAPs need only determine that S has no solution, and then halt with $C = \text{"false"}$. (For example consider the case $L = 0$ treated in Section 4.4.3, for which $k_1 * k_2$ pair-tests suffice to determine that the SAP has no solution. This case appears in general more complex and is not treated here.

Note that the above proof is not intended to say anything whatsoever about the minimum number of tests required to solve the N -queens problem, or any other PARTICULAR

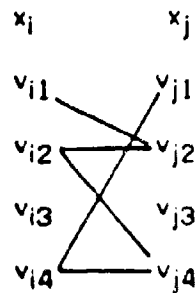
problem. Indeed there may be a clever specialized algorithm that solves the N-queens problems and no others, and can find solutions in time less than $N(N-1)/2$. Similarly the $O(N \log N)$ lower bound for sorting does not apply if we do not insist that the sorting algorithm be able to sort arbitrary inputs (permutations of $\{1,2,3,\dots,N\}$, say). If we only are concerned with sorting the particular inputs (1) , $(2,1)$, $(3,2,1)$,..., and no other inputs (e.g., $(3,1,2)$) then obviously we can devise an algorithm that runs in $O(N)$ time. Of course that algorithm may fail to sort the input $(3,1,2)$ correctly, i.e., it is not valid for all permutations, but only for a special subset. The $O(N \log N)$ bound on sorting applies to any algorithm that guarantees to sort ANY permutation correctly. Analogously, the above proof about $T_{\min}$ does not apply to algorithms that solve only a subset of all SAPs (such as the subset consisting of the N-Queens problems). Rather, it applies only to algorithms that guarantee to find solutions to any arbitrary SAP. The nature of the adversary argument is that if any such algorithm, in solving any arbitrary SAP A having N variables, executes fewer than $N(N-1)/2$ tests, then there necessarily exists another SAP B having N variables for which the algorithm will claim that B has a solution when in fact it does not, hence violating the assumption that the algorithm is valid for any arbitrary SAP.

4.5.4 Improvement to Mackworth's Version of Waltz Algorithm

The Waltz-type algorithm used in this chapter's experiments, called DEEB, reflects an improvement in efficiency over algorithm CS2 defined in Gaschnig [1974] and over AC-3 defined in [Mackworth 1977]. DEEB combines backtracking with a procedure, called DEE-O, of the generic form of "arc-consistency" algorithm that Mackworth calls AC-3 [1977] and Gaschnig [1974] calls CS-1. Mackworth [1977, p. 114] suggested certain modifications to algorithm CS-1 with the intent of improving its efficiency. DEE-O is a functionally equivalent variation of CS-1 and of AC-3 that achieves the efficiencies suggested by Mackworth and eliminates other unnecessary pair-tests as well, so that DEE-O is strictly more efficient in terms of pair-tests executed than AC-3, as we shall now show informally.

For brevity, we assume in the remainder of this section that the reader is familiar with Mackworth's argument and notation, which are used here. The following hypothetical example illustrates informally the differences between the approach of AC-3 (i.e., to distinguish the constraint relations P_{ij} and P_{ji} by distinct arc (i,j) and (j,i)) and the approach of DEE-O (i.e., to process a P_{ij} relation as a whole).

The diagram below depicts the constraint relation P_{ij} as a set of links between the candidate values of two problem variables x_i and x_j . Hypothetically, x_i and x_j could be problem variables of a SAP having other problem variables as well.



In the case depicted above, CS-1 executes the equivalent of Mackworth's function $\text{REVISE}((i,j))$, which executes 2 pair-tests (p.t.) to determine that v_{i1} is supported by v_{j2} , and then 2 p.t. to establish support for v_{i2} , then 4 p.t. to determine that v_{i3} is not supported by x_j and hence can be eliminated, followed by 1 p.t. for v_{i4} , for a total of 9 p.t. CS-1 then executes $\text{REVISE}((j,i))$, determining at a cost of 11 p.t. that all c.v.s of x_j are supported.

Mackworth correctly points out that CS-1's execution of $\text{REVISE}((j,i))$ is often superfluous, because the execution of $\text{REVISE}((i,j))$ cannot cause arc (j,i) to become "arc-inconsistent" if it is not already. For this reason Mackworth distinguishes arc ij from arc ji , knowing which to process by the search history. Therein lies the rub: since AC-3 initially puts all arcs (i,j) and their complements (j,i) on the queue Q , AC-3 executes each $\text{REVISE}((i,j))$ and $\text{REVISE}((j,i))$ at least once and for these executions AC-3 executes unnecessary pair-tests that are not executed by DEE-0.

DEE-0 executes a single procedure $\text{REVISEBOTH}((i,j))$ that has the effect of first doing a $\text{REVISE}((i,j))$, but at the same time marking those c.v.s of x_j that provide support to the c.v.s of x_i . REVISEBOTH then executes the equivalent of a $\text{REVISE}((j,i))$, modified so that only unmarked c.v.s of x_j are checked for support by x_i . Hence in the above example $\text{REVISEBOTH}((i,j))$ executes only 9 p.t. (i.e., those corresponding to AC-3's execution of arc ij , as opposed by arc ji), since all c.v.s of x_j are marked. Generalizing, in precisely the cases that the $\text{REVISE}((j,i))$ of CS-1 is superfluous due to the conditions described by Mackworth, in these same cases all c.v.s of x_j are marked, and hence $\text{REVISEBOTH}((i,j))$ executes exactly those p.t. executed by $\text{REVISE}((i,j))$. Hence DEE-0 using REVISEBOTH executes no more p.t.

than AC-3 using REVISE. Since DEE-0 executes fewer p.t. than AC-3 for the first executions of REVISE((i,j)) and REVISE((j,i)), it follows that DEE-0 executes strictly fewer pair-tests than AC-3 for all SAPs except the degenerate cases of SAPs that are arc-consistent initially.

Orthogonal to the issues just discussed, AC-3 maintains a queue of pending arcs (i,j) to REVISE, whereas CS-1 uses a triangular matrix for the same purpose, but without the FIFO discipline of the queue. Instead, CS-1 implements the "one pass" policy used by Waltz, in which the problem variables are introduced one at a time, propagating constraints until stability results after each introduction of a new problem variable. DEE-0 could use either priority policy, but in fact uses the triangular matrix mechanism of CS-1 (See Gaschnig [1974] for details.)

Add little to little and there will be a big pile.
Ovid

4.6 Conclusions and Future Experiments

The main technical conclusions summarized briefly in Section 4.0 are supported by more than 13000 distinct algorithm executions, for each of which three distinct performance values were recorded. These performance data are several orders of magnitude more numerous than previous experimental performance measurement data for any algorithm for SAPs. Knuth's article [1975] is motivated by the observation that the performance of backtracking is difficult to predict in detail a priori; data such as those given here may promote insight and constitute a set of particular values with which any predictive theory about SAP search performance must agree.

We have demonstrated that the much greater efficiency of Waltz-type algorithms over backtracking observed by Waltz and others does not extend to all SAPs, in particular not to the N-queens SAPs, random-N-queens SAPs, or ISVL SAPs tested here. The factors accounting for the differences between the present data and previous experience remain uncertain at present, but it is very plausible that the relative performances of DEEB and BACKTRACK differ depending on whether the problem to be solved has a complete consistency graph (e.g., N-Queens SAPs and random-N-Queens SAPs) or an incomplete consistency graph (e.g., Waltz' line drawing problem and map coloring problems). Several extensions of the present experiments to additional cases are proposed under item E4-1 in Appendix B.

The comparison of N-Queens SAPs with parametrically similar "random-N-Queens" SAPs represents a step toward "finer grain" analysis of algorithms research, in which the set of all problem instances is partitioned not only according to size (as in defining N = number of elements to be sorted as the size of the problem for sorting), but is sub-partitioned by other measures as well (here, by L , the degree of constraint). The general technique used here, simply to define an equivalence relation to hold on a particular set, would seem to be applicable to other sets of problem instances as well, for example those for NP-complete problems (e.g., partition the set of all instances of size N into those for which the lower bound limits apply, and those which require fewer steps). At the very least, "particular

situation problems" vs. "random problems" experiments of this sort serve to establish quantitatively how realistic model assumptions are. More speculatively, such efforts may prove useful in developing a mathematical theory about the relation between problem "structure" and algorithm performance.

The results comparing "left-to-right" c.v. ordering with random c.v. ordering (i.e., Figures 4.1.2-1, 4.1.2-2, 4.1.2-3, and 4.3-2) are less general than other results in this chapter, since the former are restricted to N-Queens SAPs, whereas other results pertain to randomly generated SAPs as well. The import of the "left-to-right" vs. random c.v. ordering comparison can be summarized as follows: (1) performance (T_f) can vary strongly with the choice of c.v. ordering (up to a factor of 498 difference was observed); (2) the heuristic of choosing a c.v. ordering randomly can give better performance than the heuristic of choosing some problem-dependent "obvious" c.v. ordering. Clearly, stronger claims than these must await the results of analogous experiments for other problems. Pending such additional results, it would seem that the assumption of random c.v. ordering for purpose of making more tractable the mathematical analysis of SAP algorithms is a realistic simplification.

It is clear that additional experiments of the sort described here are necessary to delimit the conditions (especially the range of problems) for which: (1) algorithm BACKMARK executes fewer pair-tests than the other known algorithms for SAPs; (2) $T_f(N)$ grows exponentially with N vs. subexponentially; (3) random candidate value ordering causes fewer pair-tests to be executed than does some "obvious" candidate value ordering.

Several categories of additional experiments are apparent: (1) extending the present experiments to additional sets of values of the experimental parameters (such as N and L); (2) analogous experiments with different problems (especially those having incomplete consistency graphs), (3) distinguishing more parameters in contrasting algorithms' performances on "natural" SAPs vs. parametrically similar SAPs that are generated randomly. Concise descriptions of a number of such experiments are given in Appendix B.

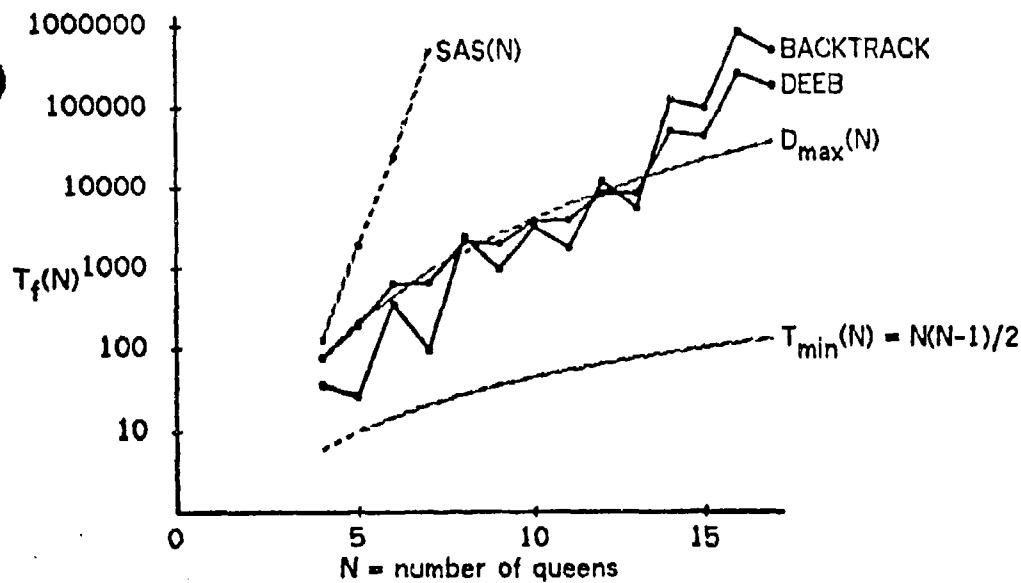


Figure 4.1.1-1 $T_f(N)$ = number of pair-tests to solve N-Queens puzzle (to find first solution)
algorithm BACKTRACK vs. Waltz-type algorithm DEEB
one algorithm execution per plotted point (solid curves)
SAS = size of assignment space

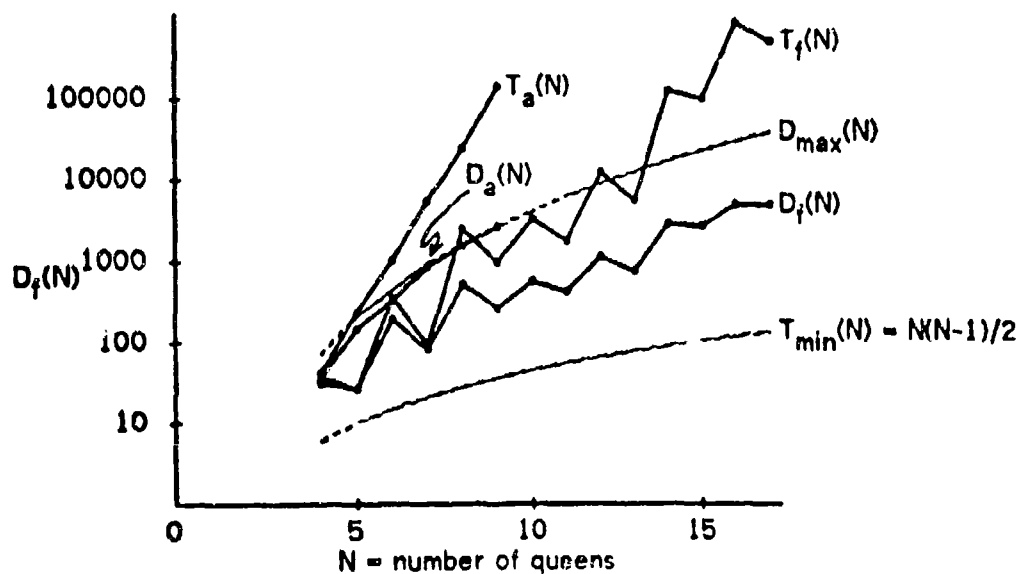


Figure 4.1.1-2 number of pair-tests (T) and number of distinct pair-tests (D) to find first solution (T_f , D_f) and all solutions (T_a , D_a)
N-Queens, BACKTRACK; 1 algorithm execution for every plotted point
 $T_f(N)$ values same as those plotted in Figure 4.1.1-1

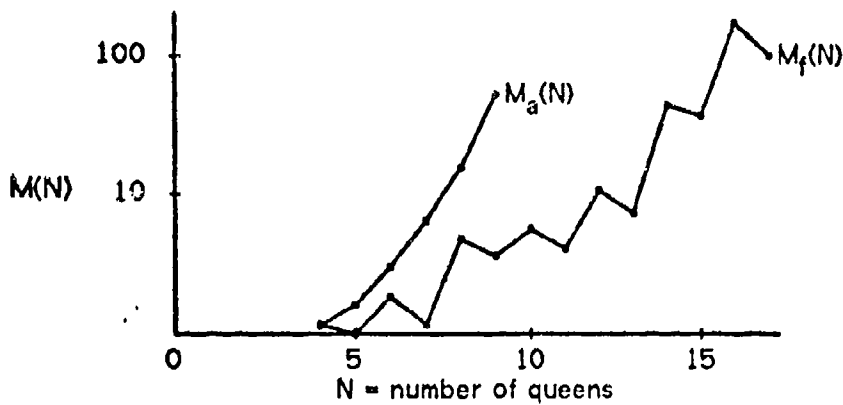


Figure 4.1.1-3 Redundancy ratio $M(N) = T(N) / D(N)$

= Total number of pair-tests executed / number of distinct pair-tests executed
 N-Queens, algorithms BACKTRACK, first solution and all solutions
 T(N) and D(N) values in computation of M(N) are those in Figure 4.1.1-2

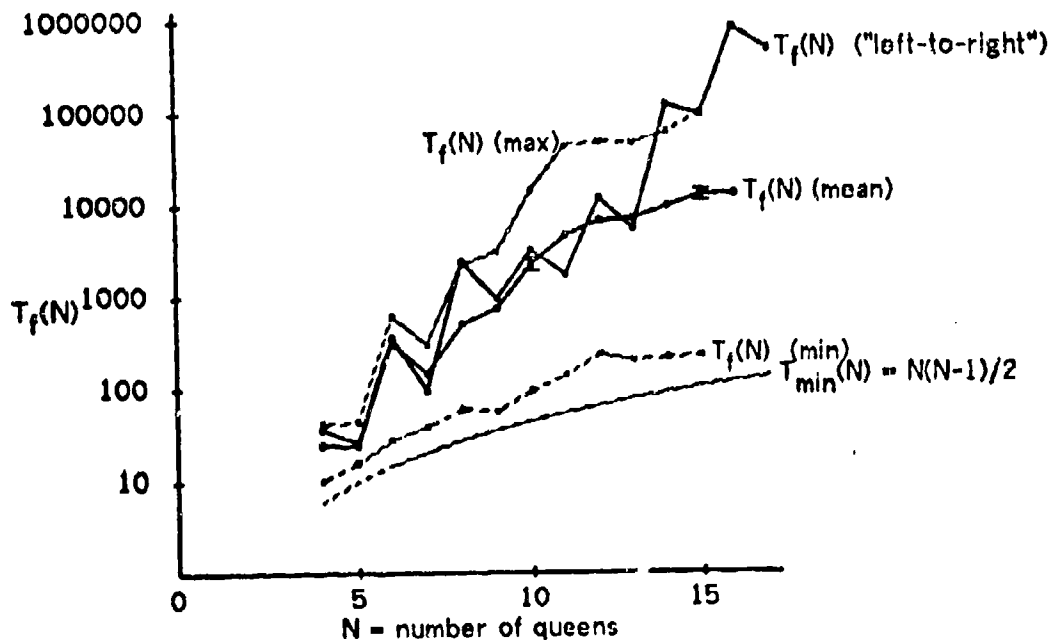


Figure 4.1.2-1 $T_f(N)$: mean, max and min values over $m(N)$ samples of random candidate value ordering, compared with $T_f(N)$ for "left-to-right" c.v. ordering
 N-Queens, algorithm BACKTRACK, first solution
 $m(N)$ algorithm executions for each value of N; $30 \leq m(N) \leq 100$ (see text)
 810 algorithm executions (a.e.) total

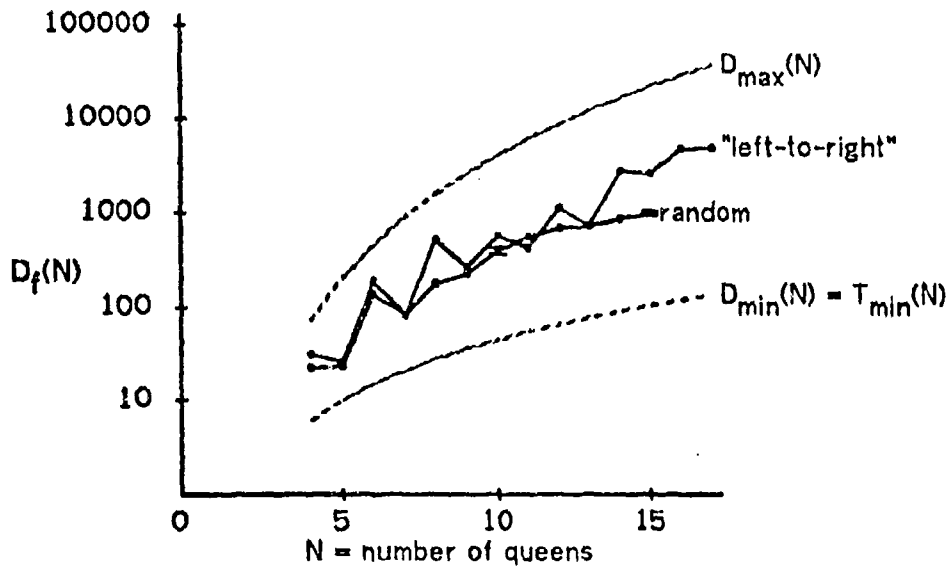


Figure 4.1.2-2 $D_f(N)$: random vs. "left-to-right" candidate value ordering N-Queens, BACKTRACK, first solution

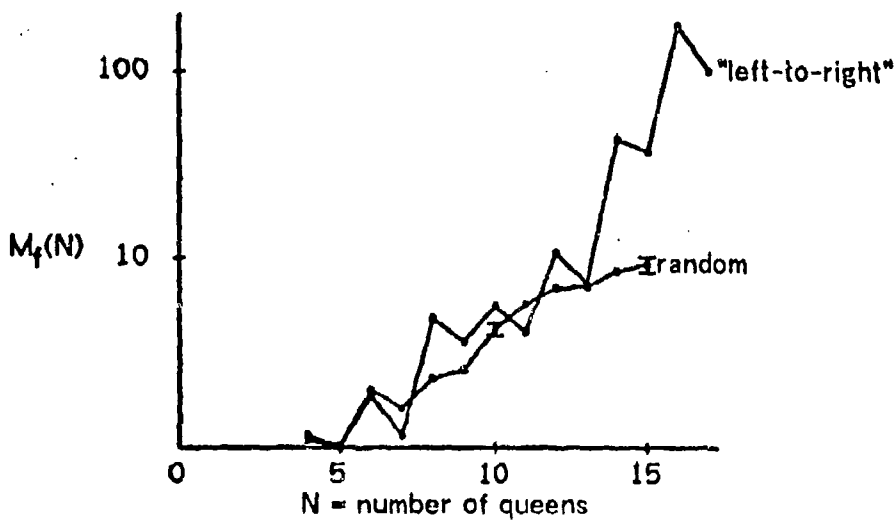


Figure 4.1.2-3 $M_f(N) = T_f(N) / D_f(N)$: random vs. "left-to-right" candidate value ordering N-Queens, algorithm BACKTRACK, first solution

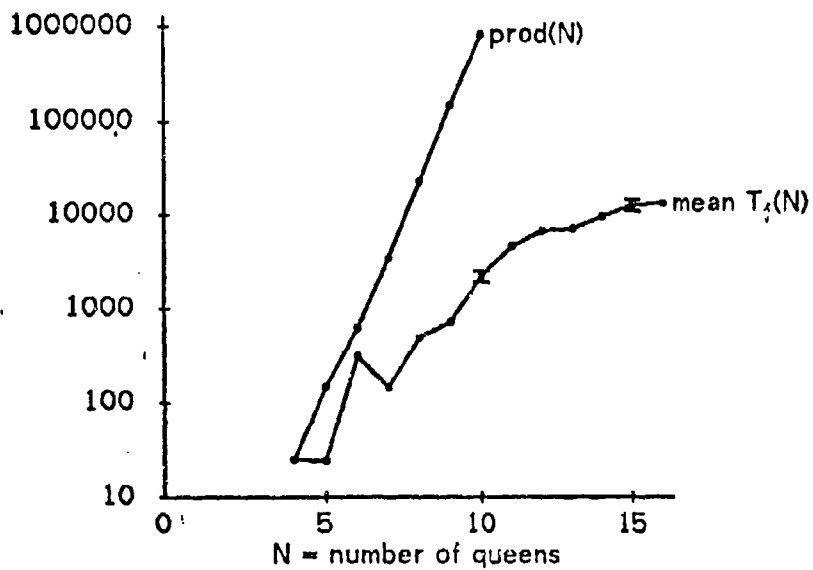


Figure 4.1.2-4 mean $T_f(N)$ compared with $\text{prod}(N) = \text{mean } T_f(N) * \text{sol}(N)$
 $\text{sol}(N)$ = number of solutions of N-queens problem.
 mean $T_f(N)$ values are those in Figure 4.1.2-1

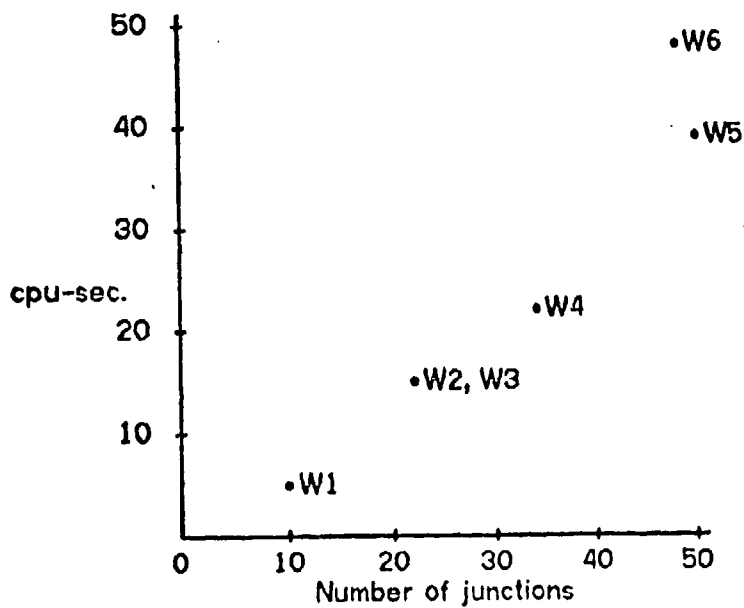


Figure 4.1.3-1 Cpu-time of Waltz' program vs. number of junctions in line drawings
 Wx denotes order of appearance of line drawing on pp. 21-22 of [Waltz 1972]

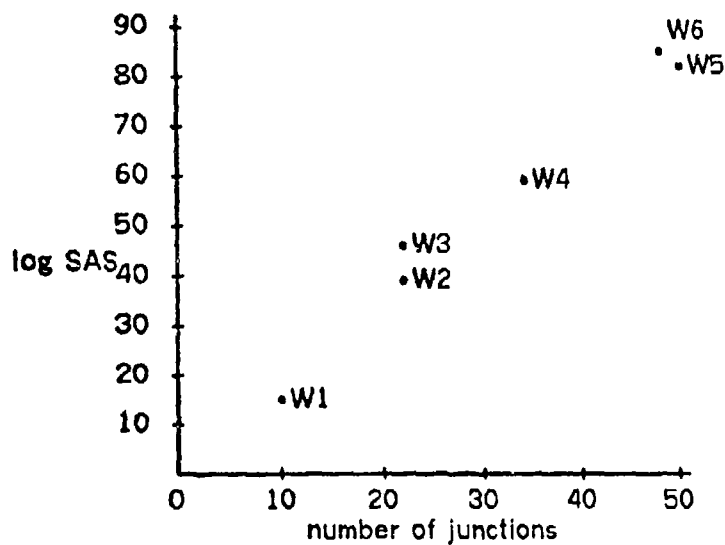


Figure 4.1.3-2 log of Size of Assignment Space (SAS) vs. number of junctions for 6 Waltz line drawings of Figure 4.1.3-1

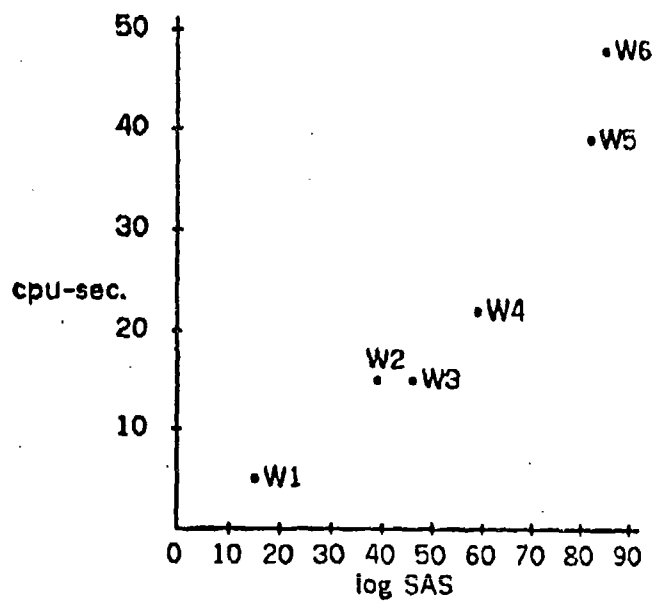


Figure 4.1.3-3 Performance of Waltz program on 6 Waltz line drawings cpu-sec. vs. log of size of assignment space (SAS)

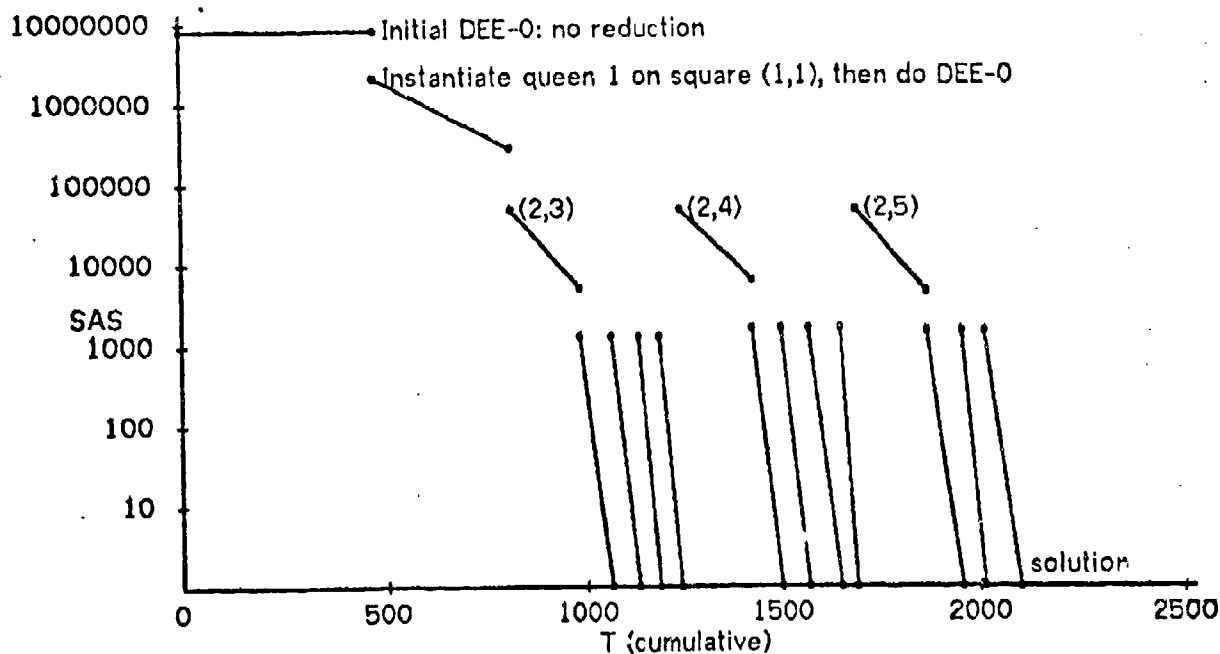


Figure 4.2.3-1 $\log SAS / T$ = reduction in state space size per unit cost
 8-Queens, DEEB, 1 sample ("left-to-right" candidate value ordering), first solution,
 $SAS = 1$ is plotted here for both contradictions and solutions.
 (All points plotted at $SAS = 1$ are contradictions except for rightmost such point)

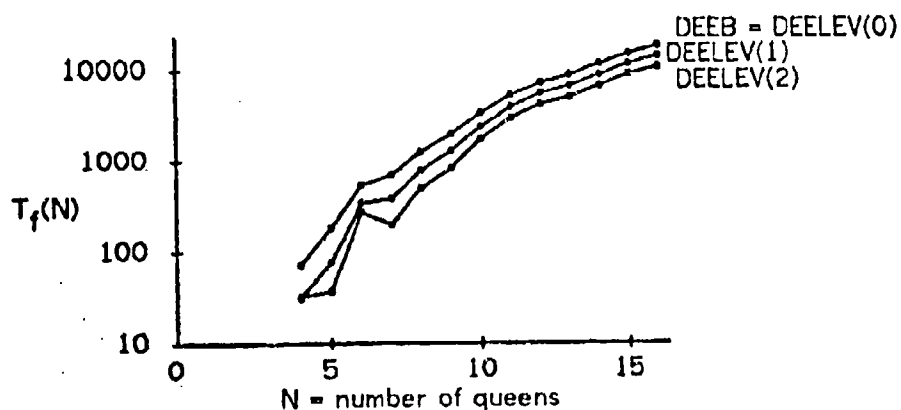


Figure 4.2.3-2 DEELEV(i): Backtrack to level i before invoking DEEB
 N -queens, first solution, random candidate value ordering
 $T_f(N)$ = mean number of pair-tests to solve N -Queens
 Same set of 810 problem instances as in Section 4.1.2.
 $3 \times 810 = 2430$ algorithm executions total

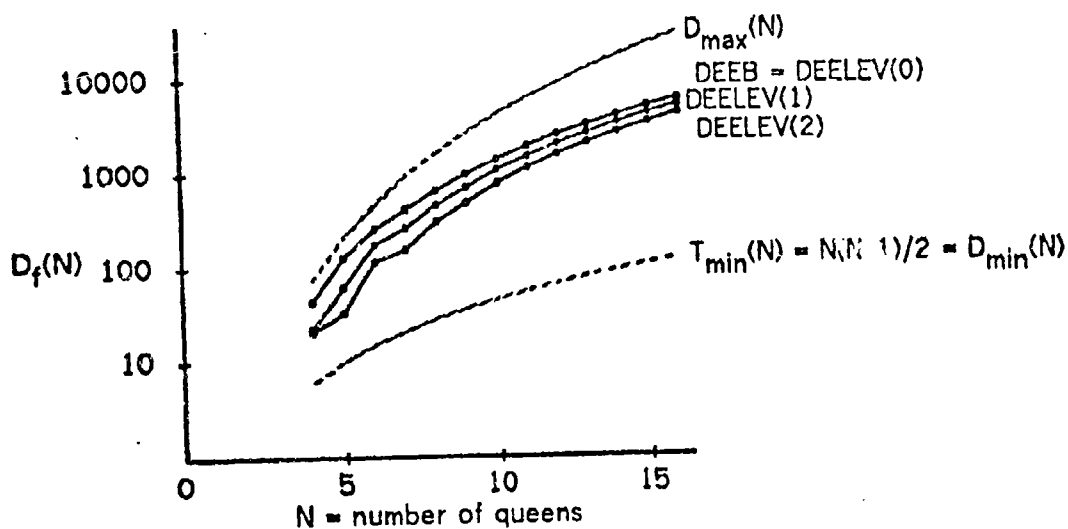


Figure 4.2.3-3 Comparison of DEELEV(i) algorithms by mean $D_f(N)$
 N-Queens, first solution, random candidate value ordering
 Data from same algorithm executions as in Figure 4.2.3-2

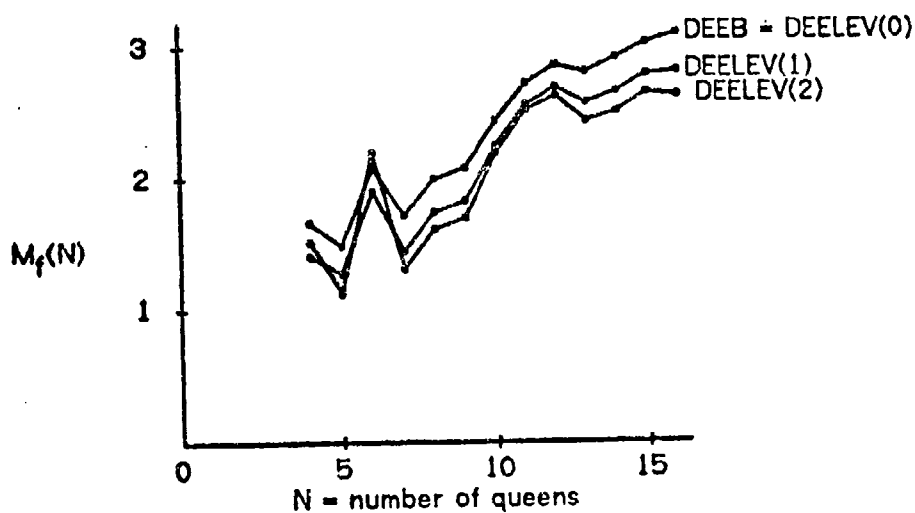


Figure 4.2.3-4 Comparison of DEELEV(i) algorithms by redundancy ratio $M_f(N) = T_f(N) / D_f(N)$
 N-Queens, first solution, random candidate value ordering
 Data from same algorithm executions as in Figure 4.2.3-2

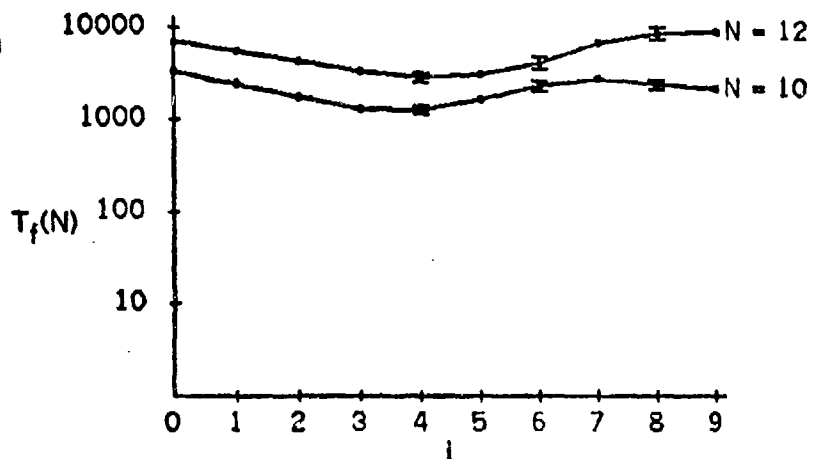


Figure 4.2.3-5 DEELEV(i) for 10-queens and 12-queens for $i = 0, 1, \dots, 9$
 first solution, random candidate value ordering
 Data for $i = 0, 1$, and 2 from same algorithm executions as in Figure 4.2.3-2
 70 algorithm executions per plotted point, 700 per algorithm, 1400 total

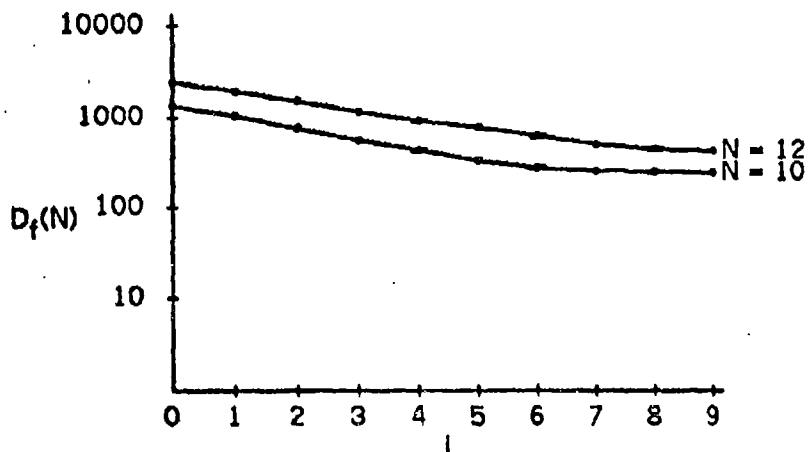


Figure 4.2.3-6 DEELEV(i) for 10-queens and 12-queens for $i = 0, 1, \dots, 9$
 first solution, random candidate value ordering
 Data from same algorithm executions as in Figure 4.2.3-5

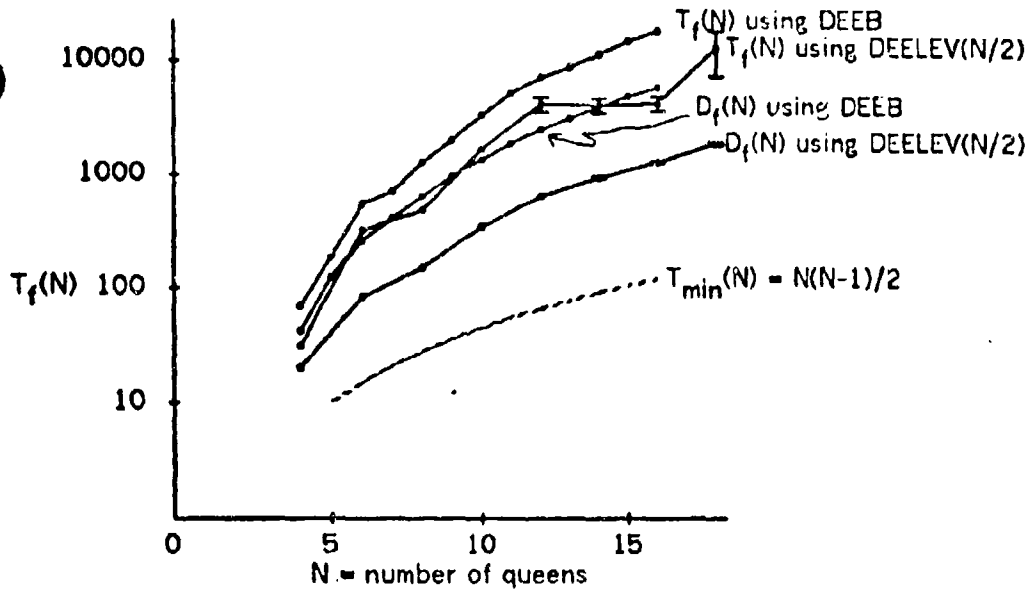


Figure 4.2.3-7 DEELEV(N/2) vs. DEEB by mean $T_f(N)$ and mean $D_f(N)$
 $T_f(N)$ = mean number of pair-tests to solve N-Queens
 N-queens, first solution, random candidate value ordering

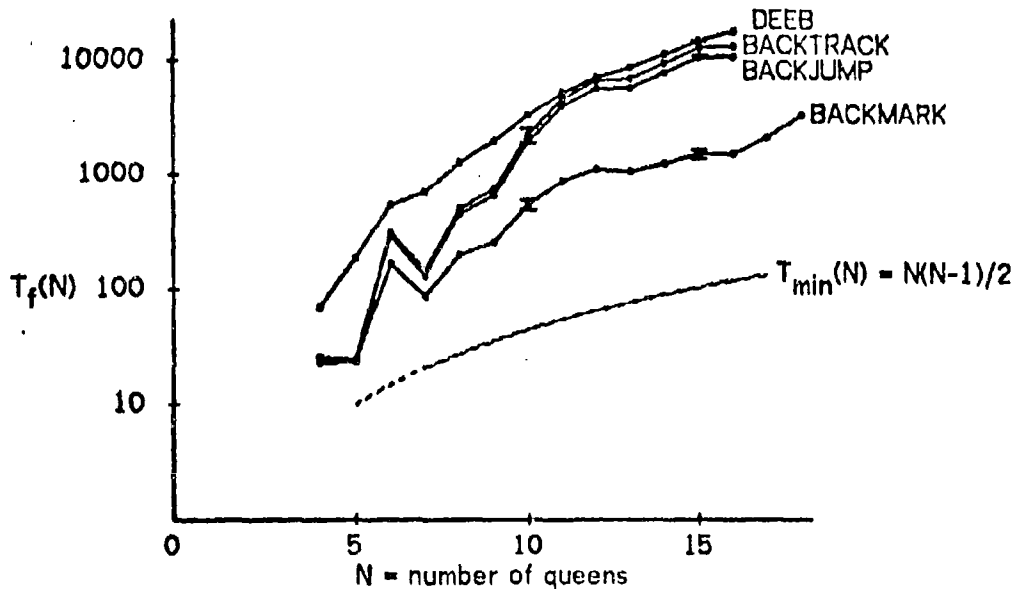


Figure 4.3-1 Comparison of algorithm performances by mean number of pair-tests
 N-queens, first solution, random candidate value ordering
 Same sample set for each algorithm (the one in Section 4.1.2).
 mean $T_f(N)$ values for BACKTRACK are those in Figure 4.1.2-1
 810-1010 algorithm executions per algorithm, 3440 algorithm executions total

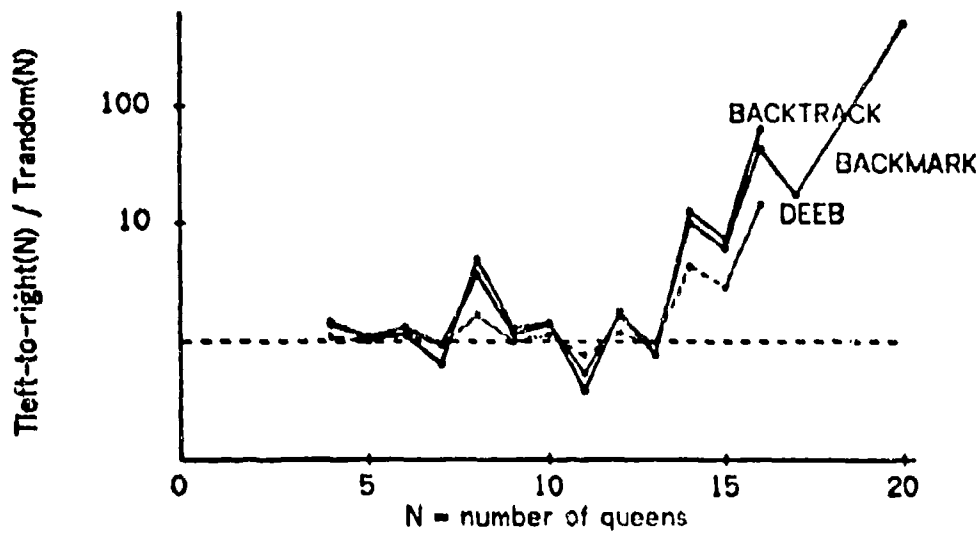


Figure 4.3-2 Ratio of $T_f(N)$ with "left-to-right" candidate value ordering to mean $T_f(N)$ with random candidate value ordering
 N-Queens, first solution, random candidate value ordering
 same set of algorithm executions as those for Figure 4.3-1
 (Values for BACKJUMP ~ values for BACKTRACK)

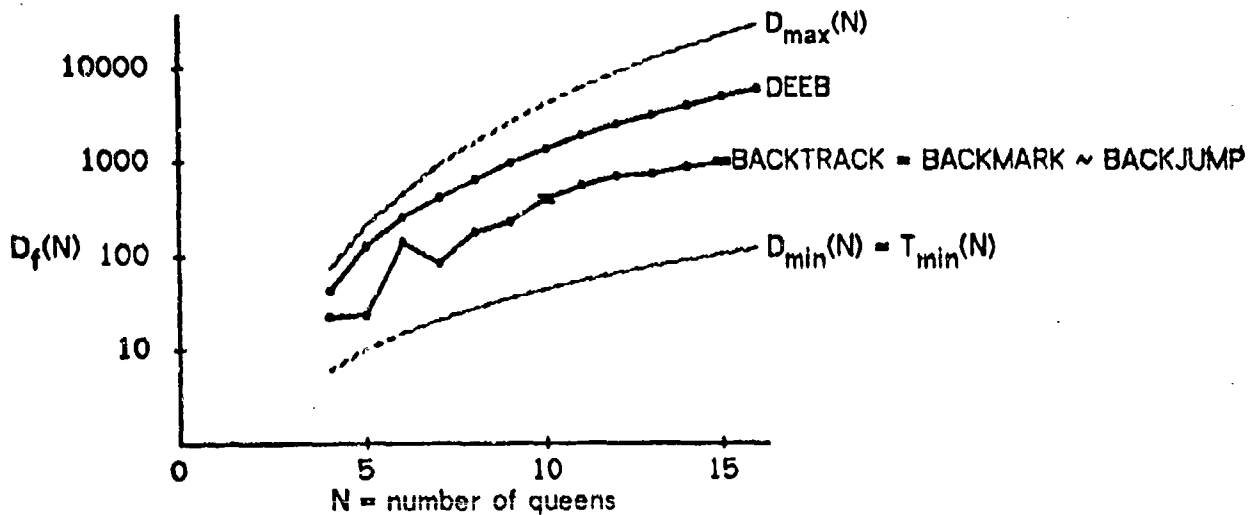


Figure 4.3-3 Algorithm comparison by mean number of distinct pair-tests
 N-Queens, first solution, random candidate value ordering
 Same set of algorithm executions as in Figure 4.3-1

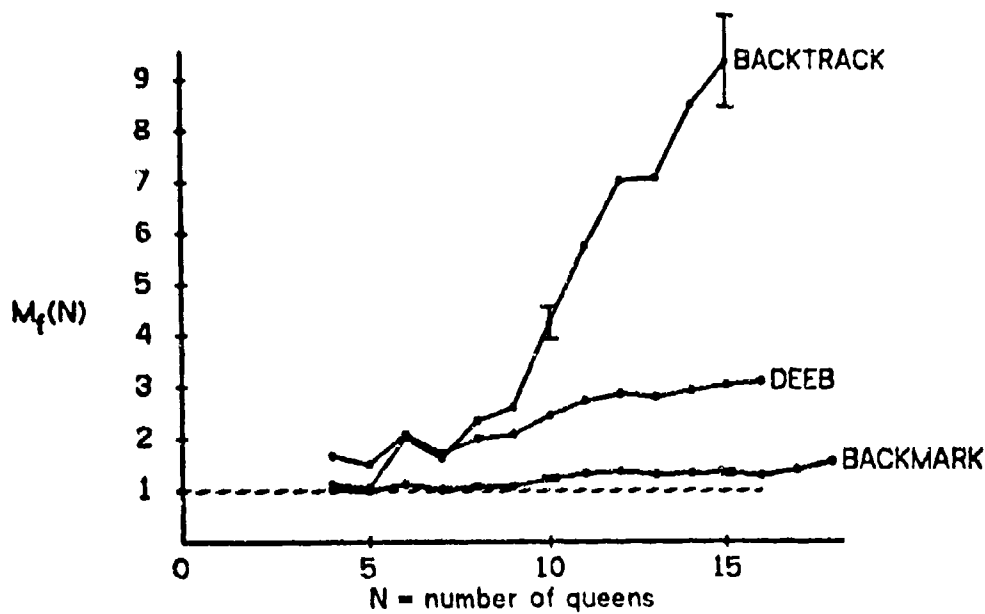


Figure 4.3-4 Algorithm comparison by mean redundancy ratio
 $M_f(N) = T_f(N) / D_f(N)$
 N-Queens, first solution, random candidate value ordering
 Same set of algorithm executions as in Figure 4.3-1

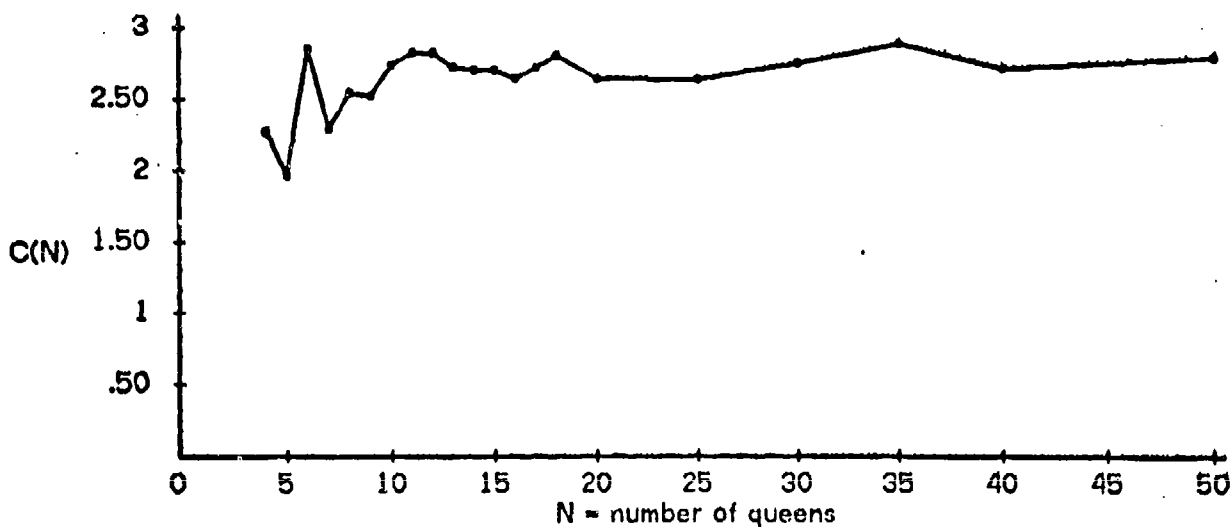


Figure 4.3-5 Growth rate of mean $T_f(N)$ using approximation mean $T_f(N) = N^C(N)$
 hence $C(N) = \log(T_f(N)) / \log(N)$
 N-Queens, algorithm BACKMARK, first solution, random candidate value ordering
 mean $T_f(N)$ values for $N < 18$ are taken from Figure 4.3-1
 50 algorithm executions per data point for $N \geq 20$, so 1310 algorithm executions total

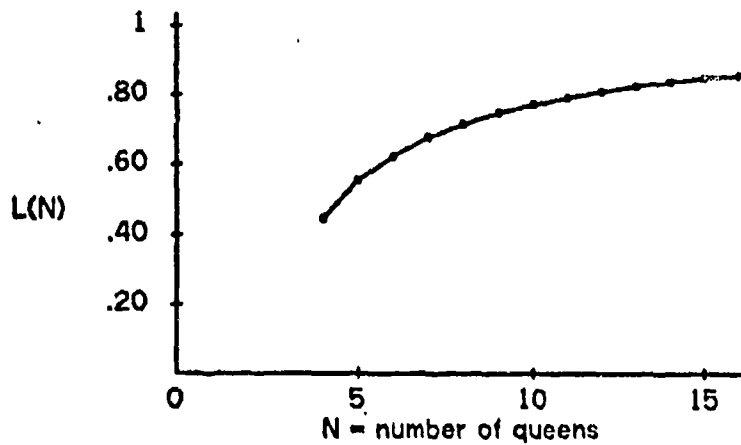


Figure 4.4.2-1 $L(N)$ = link percentage (degree of constraint) for N-Queens problem

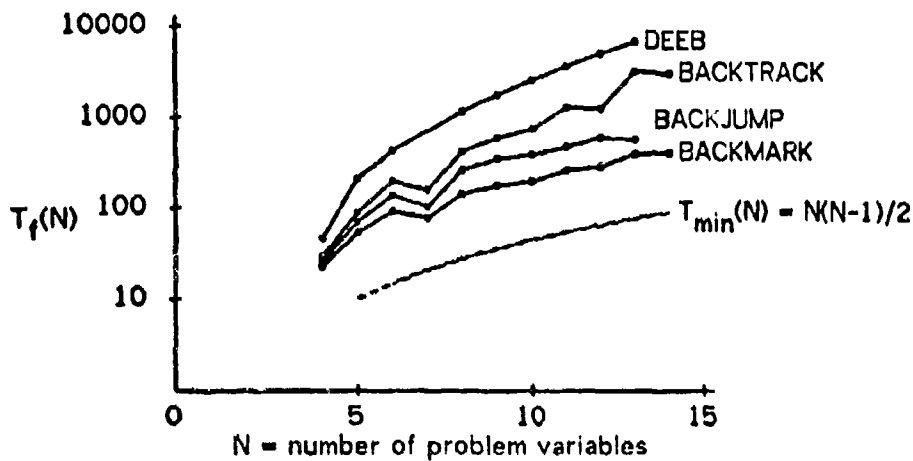


Figure 4.4.2-2 Analogous to Figure 4.3-1, but using sample set of randomly generated SAPs having same size and degree of constraint as N-Queens SAPs
 first solution, random candidate value ordering
 50-250 algorithm executions (a.e.) per data point
 850-1100 a.e. total per algorithm; 3900 a.e. total

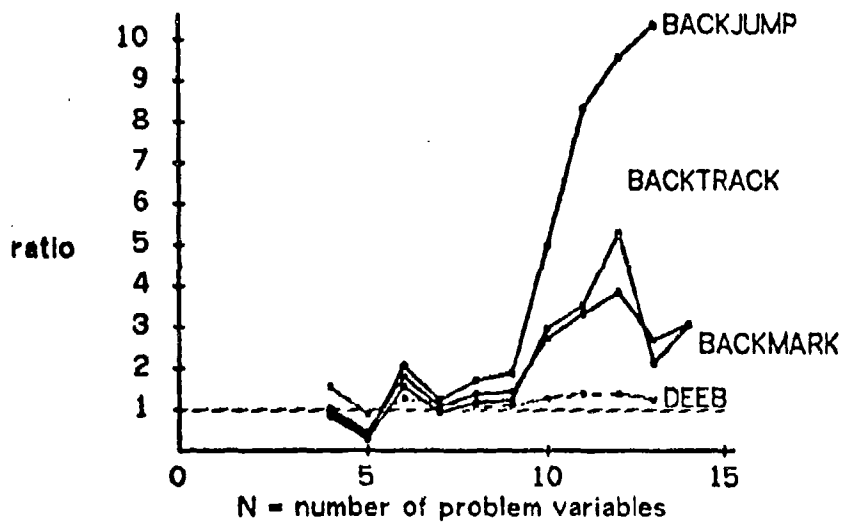


Figure 4.4.2-3 Ratio of mean $T_f(N)$ for N-Queens to mean $T_f(N)$ for "Random-N-Queens" SAPs
 Data from Figures 5.3-1 and 5.4.2-2, respectively. 3440 + 3900 = 7340 a.e. total
 Experimental data by which to distinguish "natural" SAPs from
 parametrically similar randomly generated SAPs

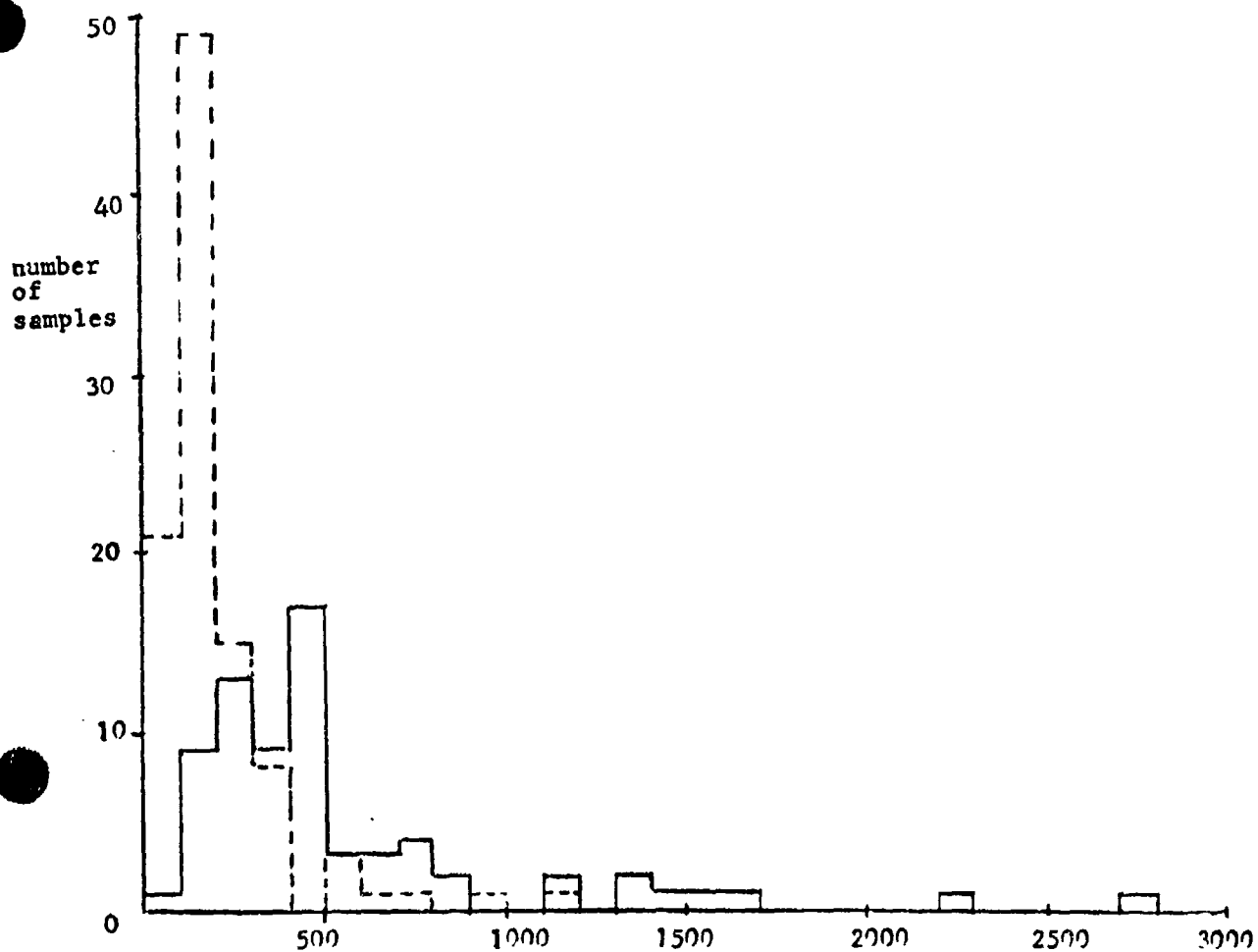


Figure 4.4.2-4 Frequency distribution of T_f values for BACKMARK applied to 70 samples of 10-Queens problem (random candidate value ordering) (solid lines), and also applied to 100 samples of "10-Random-Queens" problem (dashed lines)

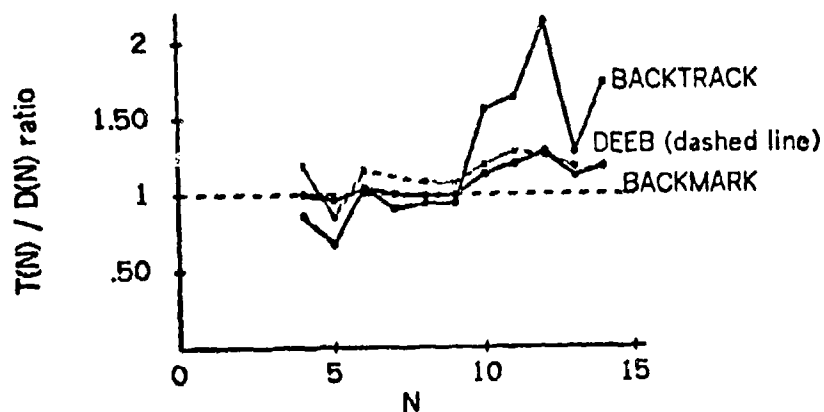


Figure 4.4.2-5 Experimental data to distinguish "natural" problems from parametrically similar i.i.d.-random problems
Ratio of $M_f(N)$ for N-Queens to $M_f(N)$ for random N-queens
first solution, random candidate value ordering

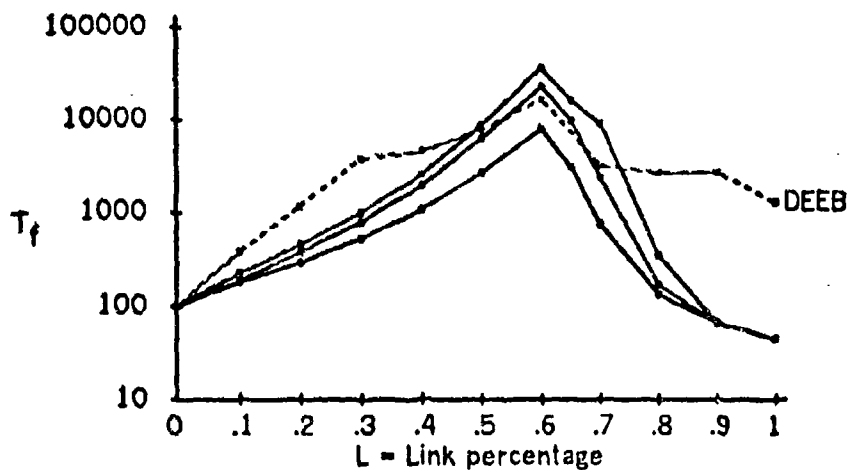


Figure 4.4.3-1 Dependence of mean number of pair-tests (T_f) on degree of constraint (L)
150 randomly generated SAPs of size $N = k_1 = 10$ for each plotted point
1350 a.e. per algorithm, 5400 a.e. total
upper solid curve: BACKTRACK; middle: BACKJUMP; lower: BACKMARK
first solution

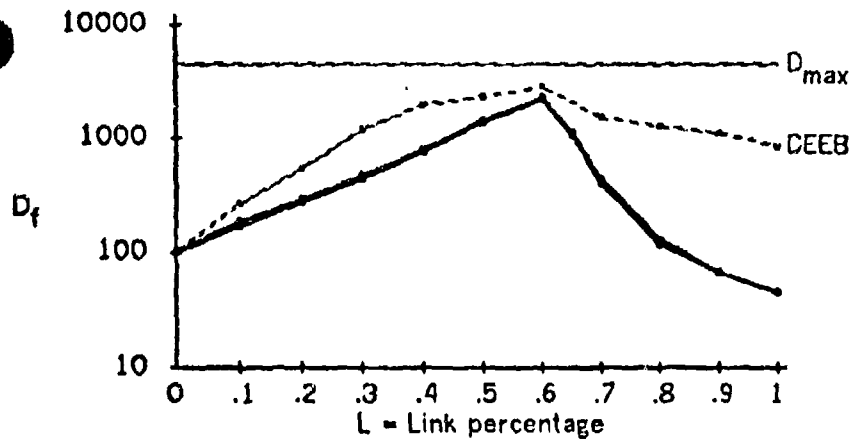


Figure 4.4.3-2 Dependence of mean number of distinct pair-tests (D_f) on L
 Same set of algorithm executions as in Figure 4.4.3-1
 Curve plots values for BACKTRACK and BACKMARK; values for BACKJUMP are almost identical.

first solution

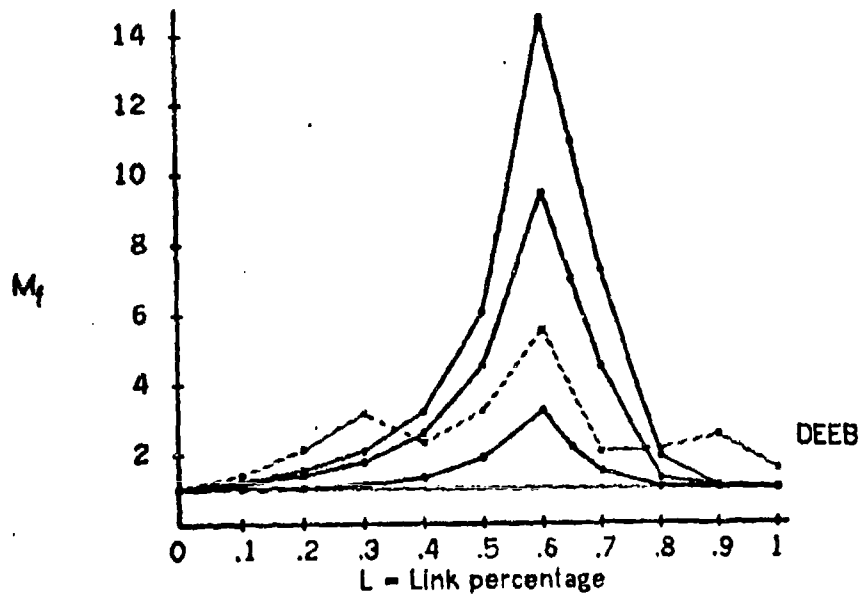


Figure 4.4.3-3 Dependence of mean redundancy ratio (M_f) on L
 Same set of algorithm executions as in Figure 4.4.3-1
 upper solid curve: BACKTRACK; middle: BACKJUMP; lower: BACKMARK
 first solution

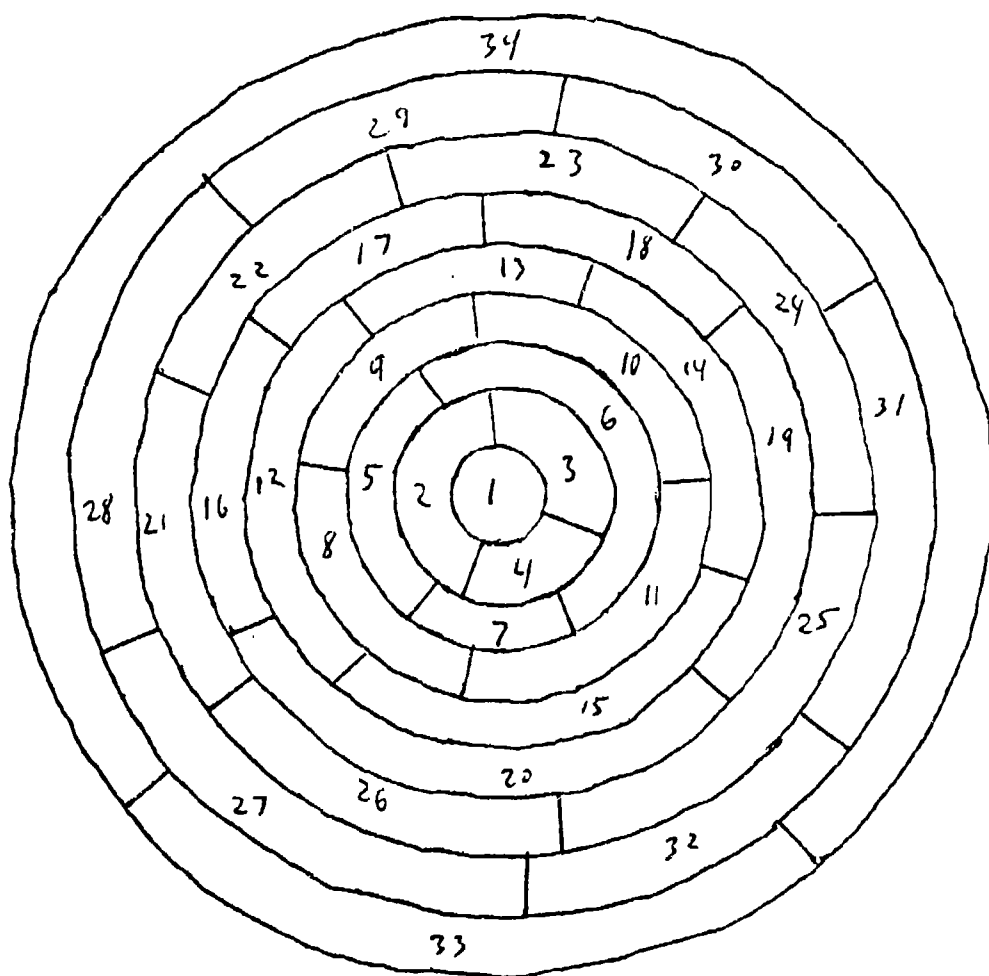


Figure 4.5.1-1 A 34 region map showing order in which regions are colored.

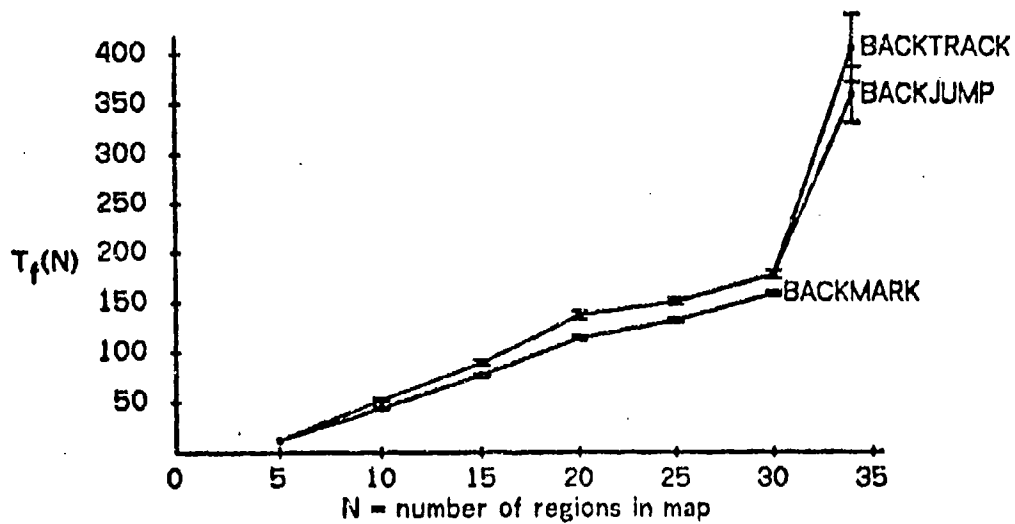


Figure 4.5.1-2 Mean number of pair-tests to 4-color a planar map
first solution, random candidate value ordering
50 algorithm executions per data point, 1000 total

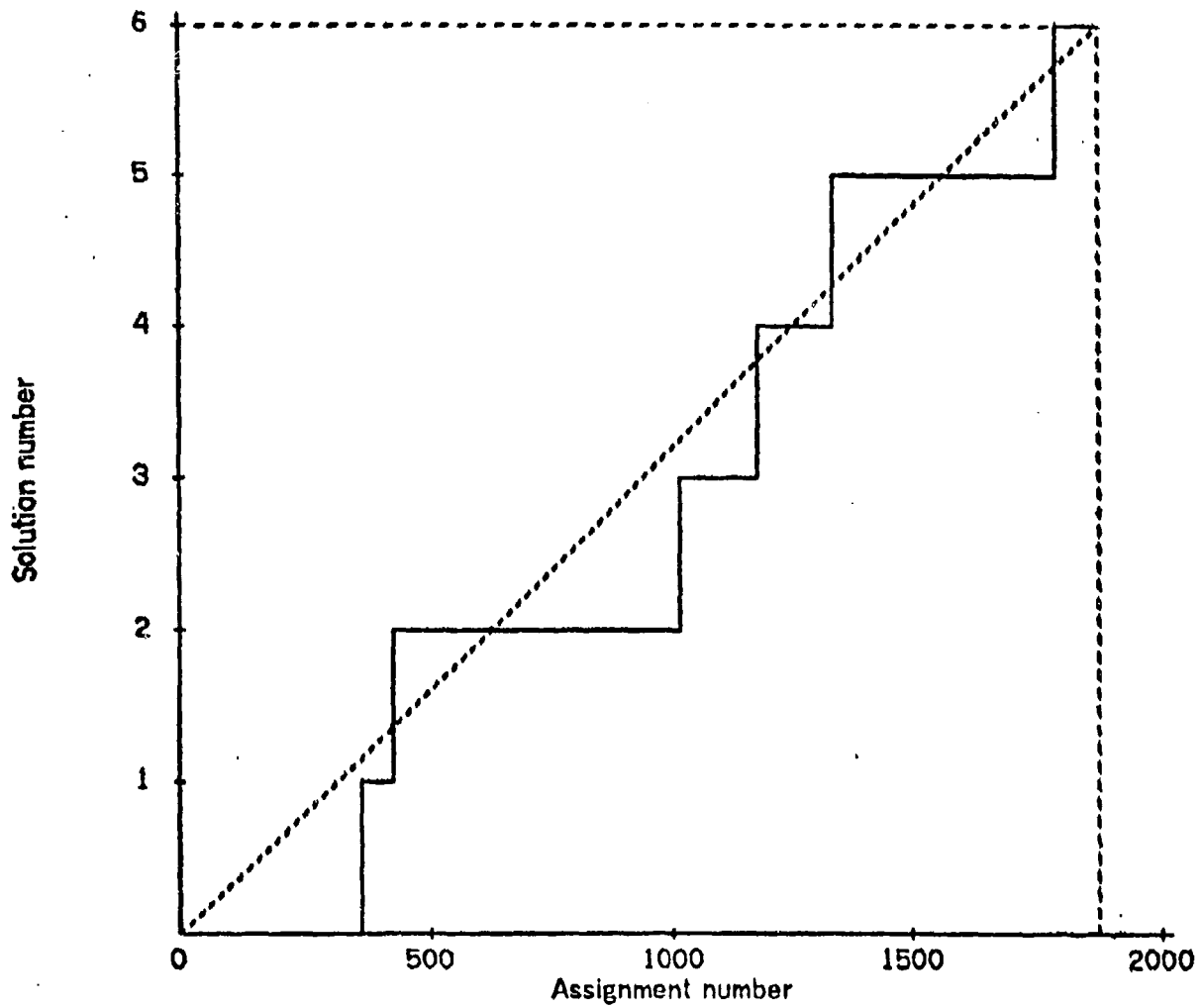


Figure 4.5.2-1 Distribution of solutions vs. assignments for 5 Queens
6 solutions, 1875 assignments
"left-to-right" candidate value ordering

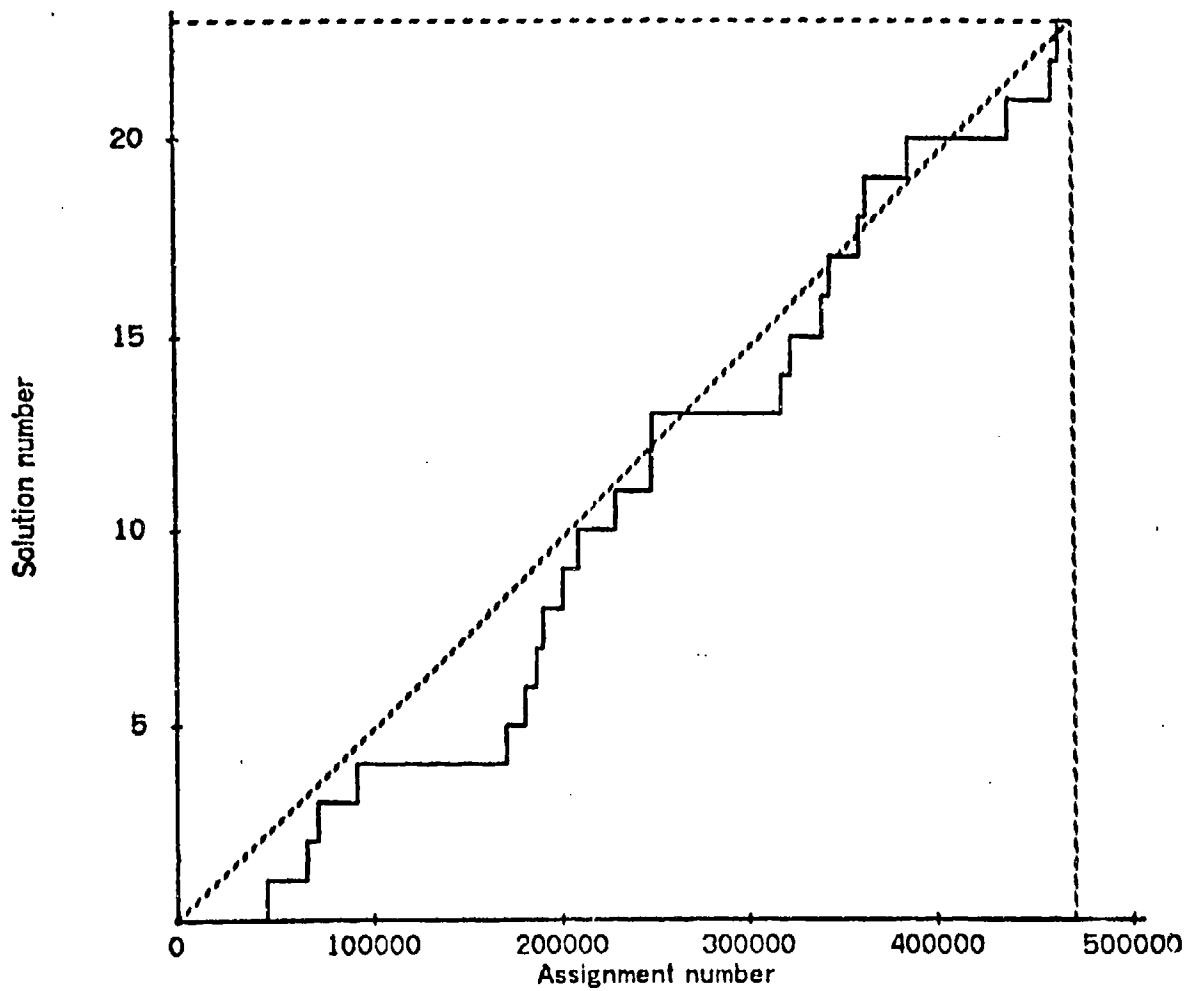


Figure 4.5.2-2 Distribution of solutions vs. assignments for 7 Queens
23 solutions, 470596 assignments
"left-to-right" candidate value ordering

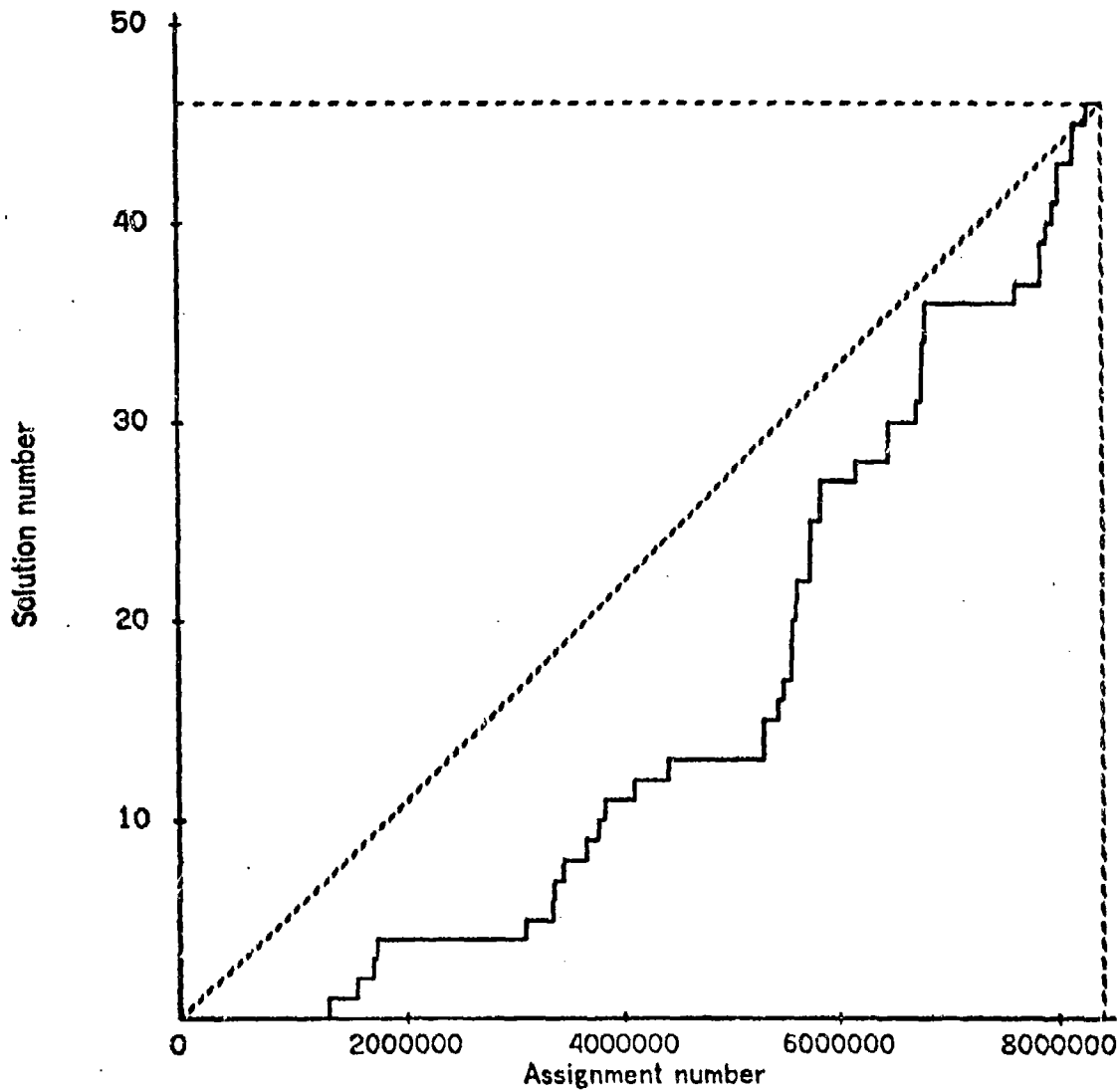


Figure 4.5.2-3 Distribution of solutions vs. assignments for 8 Queens problem
46 solutions, 8388608 assignments
"left-to-right" candidate value ordering

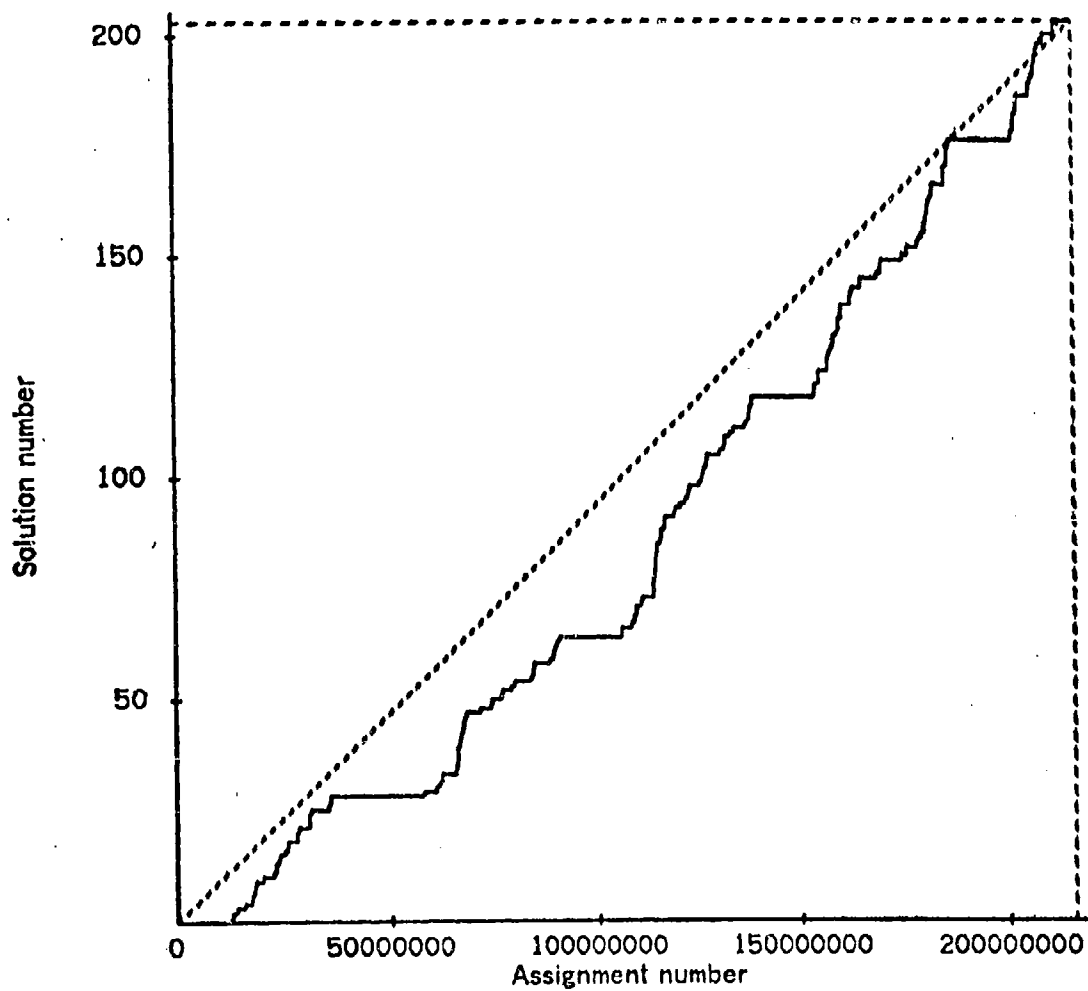


Figure 4.5.2-4 Distribution of solutions vs. assignments for 9 Queens problem
203 solutions, 215233605 assignments
"left-to-right" candidate value ordering

Chapter 5

Description of Apparatus for Search Experiments

5.0 Summary of Chapter

Effective and easily controllable tools are as important for experimental analysis as rigorous proof is for mathematical analysis. In this chapter we document the functional specifications of the computer programs that were constructed (in the SAIL language) to collect the experimental data reported in this thesis. Viewing the programs as scientific instruments (i.e., measuring devices), we assess their relevance as an aid to solving open problems in heuristic search theory. In particular, we discuss design decisions concerning the balance of five properties desirable for this application: generality, efficiency, ease and completeness of data collection, general human engineering, and modifiability.

With each program, the experimenter specifies the problem and set of problem instances to be solved, the algorithm, heuristic, and performance measurements to be collected. Some specifications are made interactively, while others are input from a previously created disk file containing problem dependent information. Several of the specification quantities may be parameterized over a range of values. After completing the specification process, the program executes without further human intervention, producing data and/or log files. An annotated trace of each program illustrates the set of options available to the user. The information provided concerning the above is intended as a reference for those who would replicate or extend the experiments described in this thesis, and for those who would implement programs having similar functions for different applications.

In this age of increasing scientific specialization, one fact constantly reasserts itself: advances in instrumentation are perhaps the most important factors in opening new fields of science.

Editorial, Science, November 20, 1977, p. 7

5.1 Issues: Generality, Efficiency, Data Collection and Analysis, Modifiability, General Human Engineering

The programs described herein were designed to satisfy certain purposes and goals, and their usefulness reflects the extent to which these goals are achieved. In general terms, the goals are to extend our theoretical understanding of state space search phenomena, but many different program designs might realize these goals. Accordingly, in this section we document the goal-motivated criteria on which the design decisions for the programs are based, and highlight a number of particular issues that arose in the design process.

The implications of the theoretical objectives of this work on program design can be expressed by three questions: What algorithm behavior can be generated? What algorithm performance measures can be defined and their values measured? How difficult is it to obtain these measurements. In ways familiar to most computer scientists (and hence not elaborated on here), the answers to these questions are determined in large part by program characteristics such as generality, efficiency, ease of data collection and analysis, modifiability, and general human engineering.

Some of these issues can be illustrated by considering Figure 4.4.2-3, as follows. (The data plotted in that figure were collected using the BKDEE program.)

Generality: had we obtained data only for the case of N-Queens SAPs and not for Random-N-Queens SAPs, Figure 4.4.2-3 could not be plotted. Had we obtained data only for algorithm DEEB or only for BACKTRACK or BACKMARK or BACKJUMP, we might draw different conclusions than those suggested by a comparison of all of those algorithms.

Efficiency: had we been able to obtain the data in Figure 4.4.2-3 only for the cases $N \leq 9$, we might draw different conclusions.

General human engineering: Figure 4.4.2-3 reflects the results of 7340 distinct algorithm executions. These required some 40 distinct executions of the BKDEE program: two sample sets of SAPs, times two groups of algorithms (DEEB on the one hand and BACKTRACK, BACKMARK, and BACKJUMP on the other), times ten values of N. A different program design permitting iteration simultaneously over several values of N, SAP definitions, and over all algorithms might have reduced the number of BKDEE executions to obtain the same data to one. As currently implemented, however, interaction between experimenter and BKDEE is succinct, and occurs in entirety before the actual execution of the experiment begins; the 40 aforementioned executions required an aggregate of about one hour of experimenter time.

Modifiability: The character of these experiments has changed and expanded with time, with experimental results suggesting other experiments of a similar nature (e.g., different performance measures or sample set of problems) and in some cases of a dissimilar nature. This tendency toward evolutionary development of the experimental apparatus necessitated a modular programming style in the present case. The concrete criteria we used in assessing success in this matter is the amount of implementation time required between the conception of a new experiment and the ability of the mechanism to execute it.

5.2 ASTAR (A* and Variants)

SAIL program ASTAR produced the experimental results reported in Chapter 2.

Various program options are specified interactively by the user, as illustrated in the following slightly edited sample trace of ASTAR and in the annotations that follow it. Line numbers are provided to the left of each line in the trace for ease of reference. Underlined text is typed by user, all other text is printed by ASTAR.

```

Record this dialogue on disk file? y
      file name = eightp.txt
001  Documentation: 2/19/78, sample trace
002  A* heuristic search system for eight puzzle
003  Trace/development/debug mode? n
004  NOTE: default K function =  $K_2$  = sum of distances, form  $F(s) = (1-w) \cdot g(s) + w \cdot K(s)$ , default  $w = .5$ 
005  Command level: Define (D), Execute (X), Auto-execute (A), Set parameters (S), Generate standard
      problem instances (G), or Exit (E)
006  Command: g

```

```

007 Change value of w? n
008 Print K values of nodes along solution path found? n
009 Change heuristic function? n
010 Set debug list? n
011 Command: g
012 Read problem instances from file? y
013 File name = input.txt
014 Iterate for several values of weighting coefficient w? n
015 w = .50000
016 Depth from 2 to 15
017 Max depth = 100
018 Max no. of nodes expanded = 40000
019 Want to see the tree? n
020 Want short output form? y
021 Collect K(n) vs h(n) statistics? n
022 Count no. of nodes expanded at each level and no. expanded after each solution path node? n
023
024 Key to data: initial state, goal state, (optional K values,) depth of goal, avg. and max run length,
025 no. nodes expanded, no. distinct nodes generated, no. of all nodes generated
026
027 Turn off tty? n
028 Depth = 2
029
030 #1: 180346275 018346275 2 2.0000 2 2 4 5
031
032 #2: 641038725 641328705 2 2.0000 2 2 6 7
033
034 #3: 714503862 714563820 2 2.0000 2 2 5 7
035
036 #4: 635241870 635201847 2 2.0000 2 2 4 5
037
038 #5: 536120487 536127408 2 2.0000 2 2 4 5
039 Average no. of nodes expanded for goals at depth 2 or greater = 2.0000
040
041 -----
042
043
044
045
046
047 Depth = 15
048
049 #1: 802571346 058362741 15 3.0526 7 58 94 152
050
051 #2: 216740583 461503278 15 4.1111 11 37 65 102
052
053 #3: 810564237 681570234 15 2.0000 10 98 160 264
054
055 #4: 517360428 032615487 15 8.0000 12 16 31 46
056
057 #5: 780532814 521360847 15 4.3750 10 35 63 97
058 Average no. of nodes expanded for goals at depth 15 or greater = 48.800
059
060 -----
061
062
063
064 Summary of no. of nodes expanded statistic

```

065

066 Goal found at specified depth or greater:

Depth	Mean	Max	Min	n Samples	Std.Dev.	Std.Dev. of Mean
068	2 2.0000	2.0000	2.0000	5	.00000	.00000
069	3 3.0000	3.0000	3.0000	5	.00000	.00000
070	4 4.0000	4.0000	4.0000	5	.00000	.00000
071	5 5.0000	5.0000	5.0000	5	.00000	.00000
072	6 6.2000	7.0000	6.0000	5	.44721	.22361
073	7 7.0000	7.0000	7.0000	5	.00000	.00000
074	8 9.4000	12.000	8.0000	5	1.9494	.97468
075	9 12.200	18.000	9.0000	5	3.7014	1.8507
076	10 14.600	23.000	10.000	5	5.1769	2.5884
077	11 14.200	17.000	11.000	5	2.2804	1.1402
078	12 23.800	34.000	13.000	5	7.8230	3.9115
079	13 23.600	38.000	14.000	5	12.260	6.1298
080	14 52.800	84.000	20.000	5	24.448	12.224
081	15 48.800	98.000	16.000	5	31.268	15.634

082 $LN(Y) = .23425 \cdot X + .35315$ 083 $Y = 1.4236 \cdot 1.2640^X$

084 Standard error of estimate (CRC Math Tables) using log y values = .15494

085 using y values = 4.8905

086 $Y = 3.3622 \cdot X + -12.393$

087 Standard error of estimate (CRC Math Tables) = 8.3999

088

089

090 Summary of mean run length statistic

091

092 Goal found at specified depth or greater:

Depth	Mean	Max	Min	n Samples	Std.Dev.	Std.Dev. of Mean
094	2 2.0000	2.0000	2.0000	5	.00000	.00000
095	3 3.0000	3.0000	3.0000	5	.00000	.00000
096	4 4.0000	4.0000	4.0000	5	.00000	.00000
097	5 5.0000	5.0000	5.0000	5	.00000	.00000
098	6 5.5000	6.0000	3.5000	5	1.1180	.55902
099	7 7.0000	7.0000	7.0000	5	.00000	.00000
100	8 6.3333	8.0000	3.6667	5	2.2852	1.1426
101	9 6.3000	9.0000	3.0000	5	2.7749	1.3874
102	10 5.9267	10.000	2.3000	5	2.7535	1.3718
103	11 6.3833	11.000	3.7500	5	2.7699	1.3849
104	12 4.3778	6.5000	2.8889	5	1.4488	.72440
105	13 5.3429	7.5000	2.0000	5	2.7448	1.3724
106	14 3.3966	6.6667	1.9091	5	1.9672	.98360
107	15 4.3077	8.0000	2.0000	5	2.2681	1.1340

108 $LN(Y) = .29133e-1 \cdot X + 1.2947$ 109 $Y = 3.6501 \cdot 1.0296^X$

110 Standard error of estimate (CRC Math Tables) using log y values = .34256

111 using y values = 1.5251

112 $Y = .95070e-1 \cdot X + 4.1111$

113 Standard error of estimate (CRC Math Tables) = 1.4690

Annotations of sample trace:

two unnumbered lines before line 001: If the user desires, ASTAR records all terminal I/O in a disk file named by the user. Call this the ASTAR log file. All numbered lines in the trace appear in the log file.

line 001: ASTAR permits the user to include one line of documentation text in the log file.

005: The A, S, and G commands are relevant to the experiments reported in Chapter 2. The program functions initiated by the G command are described in Section 2.7 (Chapter 2). The D command allows user to specify initial and goal states of 8-puzzle interactively.

006-010: No parameter values are reset in this trace; rather the trace simply indicates that the value of W can be set, that there is an option of printing K(s) values for every node on the solution path found for each search, and that any of a number (currently 5) of heuristic functions may be selected.

011: Auto-execute mode executes A* on each of a number of problem instances and for each of several values of W, according to user specifications. Program closes log file and terminates after completion.

012-013: User may specify list of problem instances interactively or read them from a disk file (as in this case).

014: If user types "yes", ASTAR prompts for initial value of W, increment, and maximum value of W.

016: These values are contained in the input file. Problem instances in input file are grouped according to distance from initial state to goal state.

017-022: User specifies search resource limits for space (and consequently for time), and specifies what performance measurements should be printed. If user types "yes" at line 021, ASTAR records K(s) values for each solution path node s for each search. The values recorded in aggregate form by incrementing values in a two dimensional array such that $M(a,b)$ = number of solution path nodes s having $h(s) = a$ and $K(s) = b$. This matrix is printed following the data printed in this trace, and max/mean/min data is printed for each dimension.

027: If user types "yes", then all subsequent text is written to the log file but not printed on the terminal (there is no further interaction with the program).

30-58: File input.tst contains five problem instances having distance to goal = $N = 2$, followed by five problem instances having $N = 3, \dots$, followed by five problem instances having $N = 15$. Measurements are omitted in this trace for $N = 3, 4, \dots, 14$.

064-081: Summary of measurements over all problem instances, grouped by N.

082-087: Least squares fit of exponential function and of linear function to mean values printed in lines 068-081.

090-113: Similar to lines 064-087, but using different performance measure.

113: Following this line in the log file (but omitted from this trace) are similar measurements for L = ratio of path length found to N . (For the conditions specified in this trace, $L = 1.0$ for all problem instances.)

As currently implemented, ASTAR executes the A^* algorithm only for instances of the 8-puzzle. ASTAR is structured, however, such that only minor modifications are required to produce a version in which the definition of an arbitrary problem graph (as defined in Chapter 2) can be input in the form of a SAIL source file, in a manner analogous to that described in Section 5.3 for program BKDEE.

5.3 BKDEE (A Family of Backtrack and Constraint Satisfaction Algorithms)

SAIL program BKDEE produced the experimental results reported in Chapter 4. The capabilities of this program reflect the objectives of that chapter: to compare each of several general algorithms by each of several performance measures, solution criteria, problem definitions, and heuristics for ordering candidate values of the problem variables.

The generality of BKDEE in performing experiments in which one of these various components can vary individually while others are fixed can be described abstractly by defining a "BKDEE experiment" as a 5-tuple (a, b, c, d, e) , where

$$a \in \alpha = \{ S \mid S \text{ is a SAP} \}$$

$$b \in \beta = \{ \text{"find first solution"}, \text{"find all solutions"} \}$$

$$c \in \gamma = \{ \text{BACKTRACK, BACKMARK, BACKJUMP, DEEB} \}$$

$$d \in \delta = \{ T, D, M \}$$

$$e \in \phi = \{ \pi \mid \pi \text{ is a candidate value ordering of } a \in \alpha \}$$

Hence the set of all such experiments is the cross product $E = \alpha \times \beta \times \gamma \times \delta \times \phi$.

BKDEE realizes a mapping from E to the natural numbers (in the case of $d = T$ or

$d = D$), or from E to the reals not less than one (in the case of $d = M$). The mean values plotted in various figures of Chapter 4 are aggregates over subsets of E , e.g., in which a , b , c , and d are fixed and e takes a number of values, or in which b , c , d , and e are fixed and a takes a number of values. We do not intend the above as a formal construction, rather as an informal and easily understandable abbreviation for identifying BKDEE experiments. It illustrates that BKDEE has, in some sense, five "dimensions" of variability.

The user defines an arbitrary satisficing assignment problem (SAP, see Definition 4-1) by supplying procedure bodies for a fixed set of five SAIL procedures that are contained in a disk file that is "required" (SAIL terminology) as a source file during compilation of BKDEE. Call this file the SAP definition file. For example, distinct SAP definition files are written for the N-Queens SAPs, the random-N-Queens SAPs, and the ISVL SAPs described in Chapter 4. The SAIL source code for the SAP definition file is one page of line printer listing in the case of N-Queens SAPs and in the case of Random-N-Queens SAPs, and is a page and a half in length in the case of the ISVL SAPs. Note that we defined three SAP definition files for the experiments in Chapter 4, whereas Chapter 4 reports results for four sample sets of SAPs (see Section 4.0). Two of these sample sets, namely N-Queens SAPs using "obvious" candidate value ordering and N-Queens SAPs using random c.v. ordering, use the same SAP definition file, the different sample sets for the experiments resulting from interactively specified program options.

Various program options are specified interactively by the user, as illustrated in the following slightly edited sample trace of BKDEE and in the annotations that follow it. Line numbers are provided to the left of each line in the trace for ease of reference. Underlined text is typed by user, all other text is printed by BKDEE.

The trace given below documents the use of BKDEE to obtain three points plotted in each of Figures 4.3-1, 4.3-3, and 4.3-4, namely those at $N = 8$ for BACKTRACK, BACKMARK, and BACKJUMP. Of these 9 values, those for BACKTRACK are shown in lines 082-084 of this trace, in the column labeled "Mean". The other of these values are omitted from this incomplete trace.

Record this dialogue on disk file? y


```

001 file name = bkqueen8
002 Documentation: 2/19/78, sample trace using 8-Queens SAP, first solution, random c.v. ordering
003 Program BKDEESAI, Version 4b (11/27/77).
004 Trace/Development/Debug mode? n
005
006 PROBLEM DEFINITION: N-Queens. Size of board = 8
007
008 Want to count distinct pair-tests? y
009   Max no. of candidate values per problem variable = 8
010 PROBLEM DEFINITION: N-Queens
011 Values of the variables: r.c = row.column, each from 1 to 8
012 queen 1:
013   1.1 1.2 1.3 1.4
014 queen 2:
015   2.1 2.2 2.3 2.4 2.5 2.6 2.7 2.8
016
017 and so on...
018
019 SAS (Size of assignment space) = 8388608, DMAX (no. of possible distinct pair-tests) = 1568
020 Key to performance measures:
021   T = no. of pair-tests executed
022   D = no. of distinct pair-tests executed
023   M = T/D = ratio of pair-tests to distinct pair-tests
024   LSAMP = fraction of distinct pair-tests executed that are true
025   Z(i) = no. of successfully instantiated candidate values (nodes) at level i of final search tree
026   BF(i) = Z(i)/Z(i-1) = Branching factor at level i of final search tree
027 Find all solutions? n
028 no. of samples = 70
029   Want to see each sample? n
030   Print full mean/min/max statistics? y
031   Want to randomize candidate value orderings for each sample? y
032   Seed for random number generator = 4567
033 Dynamically reorder values of variables by cost of subtrees beneath them? n
034 Solve for each order of instantiation? n
035 Backtrack (B) or DEE algorithms (D)? b
036 Which versions: BACKTRACK? y
037   BACKMARK? y
038   BACKJUMP? y
039   Timed BACKTRACK? n
040   Timed BACKMARK? n
041   Timed BACKJUMP? n
042 Turn off tty? n
043
044
045 Order of instantiation = 1 2 3 4 5 6 7 8
046
047
048 BACKTRACK (Backtrack Search):
049 Solution #1: 1.3 2.5 3.8 4.4 5.1 6.7 7.2 8.6
050 T = 392, D = 165, M = 2.3758 , LSAMP = .72727
051 no. of nodes at level i = 1,2,...,8: 1 1 1 3 3 6 4 1
052 BF = 1.0000 1.0000 3.0000 1.0000 2.0000 .66667 .25000
053 avg BF = 1.0000
054
055 Found 70 solutions
056 Median of 70 T (pairtest) values = 350.00 , 1st quartile = 177.50 , 3rd quartile = 676.50
057 1st decile = 84.600 9th decile = 1066.8 , Values are:

```

```

058 59 63 68 72 76 78 79 87 105 111 113 122 130 159 163 163 173 191 192 197
059 201 202 208 215 216 229 235 273 278 280 330 333 337 339 350 350 363 374
060 392 396 410 419 439 445 452 461 507 545 547 560 637 644 648 762 762 850
061 867 919 920 943 1034 1042 1061 1069 1171 1230 1520 1542 1753 2243
062
063 Median of 70 D (distinct pairtests) values = 155.00 , 1st quartile = 114.25 , 3rd quartile = 227.00
064 1st decile = 78.700 9th decile = 312.90 . Values are:
065 59 63 68 72 73 76 78 79 87 88 95 103 105 109 112 114 114 115 117 118
066 119 125 125 127 127 131 132 136 137 138 146 148 149 151 153 157 157 159
067 162 165 170 178 180 182 183 188 190 194 197 202 213 218 223 239 242 245
068 248 250 257 280 280 293 301 318 329 339 379 379 457 513
069
070 Median of 70 M = T/D values = 2.2560 , 1st quartile = 1.5837 , 3rd quartile = 3.0701
071 1st decile = 1.0000 9th decile = 3.7046 . Values are:
072 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0000 1.0893
073 1.1895 1.2614 1.3947 1.4174 1.4924 1.5039 1.5825 1.5872 1.6160 1.6378
074 1.6410 1.7029 1.7481 1.7632 1.7808 1.8151 1.8220 1.9178 1.9231 2.0292
075 2.0588 2.1733 2.1840 2.1854 2.2349 2.2770 2.2830 2.3299 2.3758 2.4317
076 2.4444 2.4928 2.5223 2.5309 2.5688 2.5899 2.6684 2.6688 2.8565 2.9096
077 3.0278 3.0423 3.0480 3.1365 3.1488 3.2690 3.3365 3.3735 3.5515 3.5565
078 3.5593 3.6283 3.6929 3.7097 3.7214 3.8490 3.9234 4.0106 4.0686 4.3723
079
080
081 Name Mean Stnddov. s.d. of mean max min
082 T: 496.34 449.36 54.097 22430 59.000
083 D: 179.37 93830 11.296 51300 59.000
084 M: 2.3247 .94554 .11383 4.3723 1.0000
085 LSAMP: .70678 .38885n-1 .46813n-2 .81013 .64116
086 nodes: 1.0000 .00000 .00000 1.0000 1.0000
087 1.1429 .42684 .51386n-1 3.0000 1.0000
088 2.1429 2.0093 .24189 11.000 1.0000
089 4.3000 4.3317 .52148 22.000 1.0000
090 6.6714 6.6370 .79900 33.000 1.0000
091 6.1286 5.1581 .62097 24.000 1.0000
092 3.3000 2.1892 .26355 10.000 1.0000
093 1.0000 .00000 .00000 1.0000 1.0000
094 BF(2): 1.1429 .42684 .51386n-1 3.0000 1.0000
095 1.7071 .93872 .11301 40.000 1.0000
096 1.8991 .74266 .89406n-1 3.5000 1.0000
097 1.6040 .52874 .63653n-1 3.0000 .66667
098 1.0525 .40335 .48558n-1 2.0000 .25000
099 .69322 .33665 .40528n-1 2.0000 .20000
100 .43035 .32777 .39459n-1 1.0000 .10000
101 Avg.BF 1.0000 .00000 .00000 1.0000 1.0000
102
103
104 BACKMARK (Backtrack Search with redundancy marking):
105 Solution #1: 1.3 2.5 3.8 4.4 5.1 6.7 7.2 8.6
106 T = 173, D = 165, M = 1.0485 , LSAMP = .72727
107 no. of nodes at level i = 1,2,...,8: 1 1 1 3 3 6 4 1
108 BF = 1.0000 1.0000 3.0000 1.0000 2.0000 .66667 .25000
109 avg.BF = 1.0000
110
111 Found 70 solutions
112 Median of 70 T (pairtest) values = 158.00 , 1st quartile = 115.00 , 3rd quartile = 253.75

```

Annotations of sample trace:

two unnumbered lines before line 001: If the user desires, BKDEE records all terminal I/O in a disk file named by the user. Call this the BKDEE log file. All numbered lines in the trace appear in the log file.

line 001: BKDEE permits the user to include one line of documentation text in the log file.

004: The trace of algorithm BACKTRACK given in Section 4.1.1 was obtained using trace mode = "yes"

006: BKDEE invokes the SAP definition procedure named PROBLEMDEFINIT, which sets certain global variables, among them the variable indicating the number of problem variables.

008-009: If the user wishes to count the number of distinct pair-tests executed, then BKDEE will allocate the requisite extra array storage in the main program block. If user types "no" in line 008, then line 009 does not appear.

010-017: BKDEE invokes the SAP definition procedure named GENVALS, which generates internal encodings for each candidate value of each problem variable, writing the encoded values in a two dimensional array called VALUES. The text displayed in these lines is incidental to this activity.

019: BKDEE computes SAS and $D_{\max}$ values according to the formulas given in Section 4.1.1 for the SAP defined in the SAP definition file.

027: User instructs BKDEE to terminate each search after finding a first solution as opposed to after finding all solutions to the SAP

028: The number of candidate value orderings. If user specifies 1 sample, then the c.v. ordering produced by GENVALS is used. Otherwise the subsequently selected algorithms are executed for each of a number of samples of the SAP (70 in this case)

029-032: If the number typed by user in line 028 is greater than 1, then BKDEE requests further information.

029: If user types "yes", then the information in the format printed in lines 049-053 for the first sample is printed for each of the 70 samples.

030: If user types "no", then the information printed in lines 056-112 and following is replaced by a summary consisting of a few lines.

031: User instructs BKDEE to randomize c.v. ordering of each sample. This is not done in the case of random-N-Queens SAPs and ISVL SAPs, for example (see below).

032: This seed is used for each of the subsequently selected algorithms, so that the random c.v. orderings are identical for each algorithm.

033-034: Options for experiments not reported in Chapter 4. Ignore them.

035-041: User selects three algorithms to be executed under identical conditions according to the preceding specifications. The procedures selected in this case contain invocations of performance measurement procedures, whose execution increases the amount of cpu-time required to execute the search. Hence if a timing of the algorithm is desired (using an internal 10 microsecond clock), the user should specify the "timed" versions of the procedures. The latter omit the aforementioned invocations.

042: If user types "yes", then all subsequent text is written to the log file but not printed on the terminal (there is no further interaction with the program).

045: This reflects the user instructions of line 034; refers to order of instantiating the problem variables.

048-101: User-specified performance measurements of algorithm BACKTRACK. Analogous information printed for algorithm BACKMARK (lines 104-112 + others not shown in trace) and BACKJUMP (not shown in trace). Note that the values printed in lines 058-061, 065-068, and 072-078 are sorted into ascending order, and hence do not reflect the chronological ordering of the samples to which they correspond. In particular, the value 201 in line 059 (say) does not necessarily correspond to the same sample as does the value 119 in line 066.

If the user gives answers other than those in trace, other program options are also possible, for example the following. If the user specifies DEE algorithms rather than backtrack-type algorithms in line 035, then at user request BKDEE will calculate exhaustively the value of L as plotted in Figure 4.4.2-1. (Note that LSAMP values in line 085 aggregate over the set of distinct pair-tests executed in a single search, whereas L values aggregate over all possible pair-tests of a given SAP.) If the user specifies all solutions in line 027, he/she is given the option of producing a data file containing the solution distribution information such as is plotted in Figures 4.5.2-1 through 4.5.2-4. The file produced by BKDEE in this case is in a format directly usable as input to the plotting program (PLOTFN, also written in SAIL).

5.4 Issues for Future Apparatus

In this section we discuss some of the limitations of the programs as presently implemented, and describe issues that may arise in extending or more fully realizing the purposes for which they were designed. Our comments here are restricted to two examples, one concerning the large volume of data generated by the programs and the varying import of those data, and the other concerned with general notions of performance measures as abstractions of algorithm behavior. From another perspective, the first example concerns balancing the components of an experimental system designed for specific purposes (i.e., a "weakest link" issue), and the other involves the program as a concrete medium in which to explore and develop new models of the problem solving process.

The ASTAR and BKDEE programs have performed several tens of thousands of algorithm executions and have reported several times that many individual performance measurements. The volume of these data makes bookkeeping and analysis tasks tedious and time-consuming if performed manually. Furthermore, the number of possible interrelations between the data grows larger with the number of data, so that conceivable patterns in groupings of the data that may provide new insight or suggest theorems may go unnoticed.

It would seem useful then at least to automate further the transfers across the interfaces between the data generation programs (ASTAR and BKDEE), a rudimentary data analysis program (DATANL), the plotting program (PLOTFN), and a rudimentary data base of experimental data that serves as input to DATANL. The objectives of such efforts would be to reduce the amount of time spent by the experimenter in manually retyping, editing, or otherwise selecting and transcribing numerical results contained in the log files generated by BKDEE and ASTAR.

The design of such bookkeeping and analysis facilities would seem from the experience reported here to be complicated by the evolutionary nature of the experimental investigations. That is, the choice of data to be plotted, to be analyzed, and the appropriate analysis techniques seems to change with time and with the insights obtained from previous experimental results. The existence now of this large body of only superficially analyzed "raw" algorithm performance data may stimulate

future efforts to resolve the above issues in a way that maximizes the usefulness of such data in further developing a precise predictive theory of state space search.

Finally, it seems useful to comment on the programs as defining a precise technical language in which meaningful theoretical statements may be expressed. In particular, viewing performance measures generally as abstractions of behavior suggests efforts to expand the set of performance measures. At least in part such efforts may be open-ended, having as their objective the discovery of new abstractions of behavior that are interesting in some sense. If such exploratory efforts are undertaken, it would seem useful that performance measures be interactively (i.e., quickly) programmable by the experimenter, rather than being fixed in the program (i.e., a "menu selection") as in the present programs. The limited results of [Gaschnig 1975] illustrate concretely this approach to high-level performance measure specification languages, and in particular suggest the appropriateness of such languages to facilitate the conception of new measures.

CHAPTER 6

Summary of Contributions and Future Work

Very few facts are able to tell their own story,
without comments to bring out their meaning.

John Stuart Mill

No generalization is wholly true, not even this
one.

Oliver Wendell Holmes

Knowledge consists in understanding the
evidence that establishes the fact, not in the
belief that it is a fact.

Charles T. Sprading

6.1 Contributions

Sections 2.0, 3.0, and 4.0 and the first parts of Sections 2.9, 3.6, and 4.6 summarize briefly the technical results that constitute the contributions of this dissertation. We shall not recapitulate those sections here, but rather will presume familiarity with them. In this section we supplement those summaries with more general comments, illustrating how the specific present results reflect the overall issues raised in Chapter 1.

6.1.1 Previous Conjectures Tested Against Hard Data

General propositions do not decide concrete
cases.

Oliver Wendell Holmes

Section 1.2.1 cited a number of general conjectures about the performances of A*, backtracking, and Waltz-type constraint satisfactions algorithm. The present experimental and analytic results variously support or disagree with these conjectures.

In the case of one such hypothesis, the experimental results in Chapter 5 comparing algorithms BACKTRACK and DEEB (a Waltz-type constraint satisfaction

algorithm) under identical conditions are nearly unanimous in the cases tested in disagreeing with Mackworth's conjecture that Waltz-type algorithms are "clearly more effective" than the backtrack algorithm for solving satisficing assignment problems. This is observed for the two sample sets of randomly generated problems we tested as well as for two sample sets of N-queens problems. These are apparently the first data reported comparing the two algorithms under identical conditions, including identical sets of problem instances and identical measures of performance.

To provide evidence for or against the proposition that DEEB performs more efficiently for highly constrained problems than for less highly constrained problems, we measured the performances of the two algorithms over a sample set of randomly generated problems that are identical in size (i.e., in the problem specification parameters $N, k_1, \dots, k_N$) but vary in degree of constraint (i.e., in the problem specification parameter L). These "identical size, varying degree of constraint" (ISVL) experiments in Section 4.4.3 show that BACKTRACK executes fewer tests than DEEB on highly constrained problems (Figures 4.4.3-1, 4.4.3-2, and 4.4.3-3). Curiously, the three of 11 values of L for which DEEB outperforms BACKTRACK are midway between the extremes of constraint (i.e., at $L = .4, .5$, and $.6$).

Of course, the present experimental results are restricted to a few case studies and hence they offer no general criteria. The experience of Waltz and others suggests that for some problems the Waltz-type algorithm is more efficient than backtrack under identical conditions. Nevertheless, the first experimental or analytic demonstration of this under identical conditions has yet to be reported. In particular, it would be interesting to contrast algorithm performance for SAPs having incomplete consistency graphs (e.g., all present results) with that for SAPs having incomplete consistency graphs (e.g., Waltz' line drawing problem or map coloring). One can readily speculate that this problem characteristic determines in many cases which of the two algorithms is the more efficient. Future experiment or analysis will tell whether this is true always, sometimes, or never.

In another case, the data reported in Chapter 4 do not support Mackworth's conjecture that the number of steps executed by BACKTRACK grows exponentially with the number of problem variables. Of course, one cannot infer asymptotic behavior of a function from a finite number of values, so the data serve only to affect

our confidence in the validity of this hypothesis for the cases tested. Nevertheless, the results of analysis must be consistent with those of experiment, such as those in Figure 4.3-5, which show for BACKMARK that the mean number of pair-tests (i.e., mean $T_f(N)$) for N-Queens problems is closely approximated by the formula $N^{2.75}$, for N up to 50 and with the exception of small N.

Similarly, Nilsson, Pohl, and Vanderbrug speculated that increasing the value of the weighting parameter W in A* search will cause a decrease in the number of nodes expanded. The experimental data in Chapter 2 confirm this effect if N , the distance to the goal, is large, but the data also show that for smaller N , increasing W beyond a certain value $W_{N,K}$ (i.e., the value depends in a certain way on the value of N and on the heuristic function K) actually increases the number of nodes expanded (Figures 2.3-1, 2.3-2, 2.3-3). This effect observes a certain pattern described in Section 2.3. Section 6.1.2 describes an attempt to exploit this phenomenon as the basis for a new, potentially more efficient version of A* that permits W to vary dynamically during a single search.

The latter phenomena (which we refer to by the name "Crossover N decreases with increasing W ") appear new. Hence the data show in this case that the previous conjectures are in fact somewhat simplistic, i.e., they don't distinguish enough cases explicitly and they don't predict individual numbers. However, at the time of the conjectures it would have been difficult to be more explicit, since the prior experimental data that suggested the conjectures were limited.

Addressing this "optimal W " hypothesis analytically, we proved in Chapter 3 (Theorem 3.4-1) that Pohl's results that $W=0.5$ gives better worst case performance than $W=1.0$ are not peculiar to the set of "constant absolute error" heuristic functions he considered, but generalize to the broad class of "IM-tight-underestimating" heuristic functions. We showed in particular for that class that $W=0.5$ gives better performance than any other value of W , not just better than $W=1.0$. For another class of "linearly bounded" or "constant relative error" heuristic functions (this class overlaps the IM-tight-underestimating class) we distinguished analytically the locus of heuristic functions for which $W=0.5$ gives better performance than $W = 1.0$ from that for which the opposite is true.

Here then are instances in which we applied experiments and analysis to provide additional answers to open questions posed in the literature. In general, the present results underscore the need to define such conjectures in more precise terms, and to obtain much more extensive results in answering them. Note that we make no claims about algorithm performance except for the cases tested in the dissertation.

6.1.2 Practical Applications: New Algorithms and Practical Predictions

One measure of the success of a theoretical endeavor is its ability to spawn useful practical applications. Here we cite several from among the present results.

Based on insights obtained by detailed inspection of the performances of the backtrack and Waltz-type algorithms for the satisficing assignment problems defined in Chapter 4, we have devised three new general algorithms, BACKMARK, BACKJUMP, and DEELEV. We provided rigorous and extensive experimental evidence comparing the performances of BACKMARK and BACKJUMP with the backtrack algorithm and the Waltz-type algorithm, to which they are functionally equivalent. The algorithms are compared under identical conditions. We determined how the performance of DEELEV varies as a function of its control policy parameter l .

The experimental results favor BACKMARK highly over the other algorithms tested. We summarize this evidence here. Up to a factor of ten improvement in computation speed over the backtrack algorithm was observed, and the improvement was observed over a variety of disparate problems, both N-Queens problems and randomly generated problems. BACKMARK is very fast in cpu-time: it finds solutions to the 50-queens problem (having a search space of about 10^{84}) at the average rate of one per 9 cpu seconds on a DEC KL-10. The additional storage required by BACKMARK over that of the backtrack algorithm is modest, comparable to that required to encode the problem instance. Since BACKMARK never performs more pair-tests than the backtrack algorithm and sometimes a factor of ten fewer, BACKMARK is preferred to the latter for all SAPs. Hence the present evidence indicates strongly the desirability of further experimental and analytic investigation of the computational properties of BACKMARK.

The basis for a third new algorithm, DEELEV(i), was uncovered by a detailed performance measurement experiment (Figure 4.2.3-1) that showed the source of some of the inefficiency of the DEEB algorithm: efficiency of individual invocations of the DEE-0 subprocedure (i.e., the current version of the original Waltz algorithm) decreases with depth in the search tree.

We also devised (Section 4.5.4) a new version of the underlying DEE-0 procedure that is more efficient than previous versions [Mackworth 1977, p. 114].

In another case, very extensive experiments have uncovered new phenomena (i.e., the "crossover N decreases with increasing W " phenomena described in Section 6.1.1) that hypothetical future algorithms may attempt to exploit, namely the potential for increasing efficiency of A^* search by varying the value of W dynamically during the search instead of treating W as a constant during the search. This possibility is described in more detail under item E2-2 in Appendix B.

Yet another instance in which algorithm performance results suggest directions in which to seek (or in which to avoid seeking) new algorithms arises in Chapter 4 by what might be called the " $D_{\max}$ barrier". If $T_f > D_{\max}$ for a given algorithm, then better performance may be obtained by an algorithm that executes each pair-test at most once (i.e., $M=1$) but may execute as many as $D_{\max}$ pair-tests. Such a hypothetical algorithm would presumably attempt to exploit a time-space tradeoff. By demonstrating that BACKMARK algorithm is nearly optimal by the redundancy ratio measure M in the cases tested, we have presented evidence that such a hypothetical algorithm will not necessarily be much more efficient than BACKMARK.

The experimental data also support certain generalizations that may facilitate design choices made in practice. For example, the experimental results in Chapter 2 showing how cost varies with the value of W suggest always using $W = 1$ when solving problems for which minimizing the length of the solution path found is not absolutely necessary. The choice $W = 1$ minimizes the number of nodes expanded, and this is observed for all three heuristic functions tested. (Of course, whether this generalizes to other problems remains unknown.

Similarly, the algorithm comparisons in Chapter 4 suggest using algorithm BACKMARK to solve SAPs.

We cite yet another instance in which the present data can guide the practitioner: the factor of 791 range in performance observed in the "identical size, varying degree of constraint" (ISVL) experiments in Section 4.4.3 suggest that efforts to introduce additional constraint may be rewarded by greatly improved efficiency for each of the algorithms (in terms of "moving down from the peak" in Figures 4.4.3-1, 4.4.3-2, and 4.4.3-3). In these cases ($L < .6$), introducing additional constraint (with the effect of decreasing the value of L) can allow the resulting version of the problem to be solved faster than the original version. On the other hand, these data also show that some problems that are relatively unconstrained ($L > .6$) might be solved faster if the constraint could somehow be loosened further (with the effect of increasing the value of L). It would seem that introducing additional constraint to a given problem might be an easier technique to apply in practice than loosening the constraint of a given problem, for the following reason. In devising a variation S' of a given problem S , one desires that solutions found for S' are also solutions for S . This is guaranteed when one imposes additional constraint on the given problem (i.e., such that $P'_{ij} \subseteq P_{ij}$ for all i and j , in the terminology of Definition 4.1). For example, let the "8-Queens-Knights" problem be defined like the 8-Queens problem, except that the pieces function either as queens or as knights. Hence the 8-Queens-Knights problem is a more constrained version of the 8-Queens problem, since each piece attacks more squares in the former than in the latter. Hence any solution for the 8-Queens-Knights problem is also a solution for the 8-Queens problem. The unimodal peak in the ISVL data suggests that whether introducing additional constraint in such manner results in a problem easier or harder to solve than the given problem depends on whether the degree of constraint of the given problem lies to the right or to the left of the value of L at which cost peaks. On the other hand, solutions to the 8-Rooks problem (defined analogously) are not necessarily solutions to the 8-Queens problem. Generalizing, loosening the constraints in a given problem in such manner (i.e., such that $P_{ij} \subseteq P'_{ij}$ for all i and j) permits no guarantee that a solution for the less constrained problem is also a solution for the given problem.

Note that the present evidence is consistent with these choices or practical predictions, but makes no guarantees for other cases yet untested. Hence we term these "practical" predictions.

6.1.3 A "Successive Set Partitioning" Approach to Problem "Structure"

As discussed at the beginning of Section 4.4, the practical usefulness of a mathematical analysis of an algorithm (such as the backtrack algorithm) valid for a broadly defined class of disparate problems (such as satisficing assignment problems) depends on the assumptions imposed for tractability. Results based on probabilistic assumptions defining parameterized ensembles of problems treated as random variables may yield predictions that are inaccurate when applied to a particular member of the ensemble. Fortunately, experiment can serve to compare algorithm performance for an individual problem (e.g., the 8-Queens problem) with that for a sample taken from the ensemble to which it belongs (e.g., the "random-8-Queens" ensemble), and hence give evidence as to whether the individual problem is typical of the ensemble to which it belongs. This is the subject of Section 4.4.2.

Section 4.4.2 compares N-queens problems to parametrically similar randomly generated problems by the criteria of algorithm performance. The results (Figure 4.4.2-3) are intriguing. The experimental data for $N < 10$ show a relatively small difference in algorithm performance between N-Queens problems and "random-N-Queens" problems. For $10 \leq N \leq 14$ there is a large difference. Similarly, the data for algorithm DEEB suggests a small difference over the entire range of N tested, whereas for the backtrack-type algorithms a large difference occurs for $N \geq 10$. Hence these results indicate in this case that the degree of similarity between "particular structure" problems and "random structure" problems depends on the size of the problem, and on the algorithm by which the comparison is made. The differences among the performances of the algorithms demonstrate a sense in which "structure" lies in the eye of the beholder, so to speak, at least as observed using this "successive set partitioning" technique to specify experiments.

These data raise questions: How can we explain the fact that the backtrack-type algorithms require more steps to solve N-queens problems than to solve parametrically similar randomly generated problems? Granted that the backtrack-type algorithms are valid for all satisficing assignment problems and are not specialized for N-Queens problems, one might nevertheless suppose that the existence of a high degree of regularity in a problem at least would not hinder an attempt to solve it, even using a non-specialized algorithm. What accounts for the visually apparent discontinuity

between $N = 9$ and $N = 10$? How can one explain the fact that the magnitude of the difference between "particular" and "random" problems varies with the algorithm used to solve them? In particular, is the Waltz-type constraint satisfaction algorithm DEEB generally insensitive to the existence of regularity in a problem, as the data would seem to suggest? This must be true for some problems and not for others; finding a criterion characterizing the dichotomy is an obvious open problem.

Incidentally, these data demonstrate quite clearly how changing the conditions, even the numerical parameters (e.g., $N=9$ vs. $N=10$), of an experiment can alter the outcome drastically. The implication for drawing general conclusions from limited hard data is clear. What would be shown by data for $N = 15$, 20, or 50 analogous to those plotted in Figure 4.4.2-3 remains an open question.

Note that the "successive set partitioning" technique is defined in quite general set theoretic terms. Section 6.2.3 proposes extensions of the SAP experiments by introducing additional problem specification parameters, with the effect of inducing additional partitions on the set of all SAPs.

6.1.4 Analysis of the DEBET "Arbitrary Heuristic" Model of Worst Case A* Tree Search

We saw in Chapter 3 that Pohl and others modeled the heuristics usable by A* by two parameters bounding the distance-estimate values of a heuristic function, both parameters having scalar values. In Chapter 3 we extended that model to allow estimate-bounding parameters that are themselves arbitrary functions from the natural numbers to the non-negative reals. Within this extended model, which we call the DEBET model, we derived formulas for the number of nodes expanded by A* in the worst case. The list of independent variables in these formulas includes the aforementioned two functions as control policy parameters, as well as another control policy parameter representing the relative weight given the heuristic and non-heuristic terms in the evaluation function, as well as problem specification parameters representing the depth and width of the tree that is searched.

Hence our analytic results on worst case A* performance constitute a substantial

relaxation of the restrictive "constant absolute error" assumptions of Pohl, Vanderbrug, and others. Not only is the DEBET model general enough to include within its scope heuristic functions that occur in practice, and hence permit experimental tests of its predictive ability in non-trivial case studies (e.g., the 8-puzzle test in Section 3.5), but we have proved theorems and derived formulas that have simple enough statements to permit insight. Theorem 3.3-2 is one such example, stating that cost depends in a certain sort of exponential way on the relative error in the heuristic function's distance estimates. Theorems 3.2-5, 3.2-6, 3.2-7, 3.3-3, and 3.3-4 constitute another example, showing conditions for which cost (i.e., $XWORST(N)$) grows monotonically with relative error in the heuristic function's distance estimates.

Possibly noteworthy from an analytic technique point of view is the manner in which the analysis proceeds in stages, mapping from the set of K functions to the set of $KMIN$ and $KMAX$ functions to the set of relative error $(\delta(i))$ functions to the set of $YMAX$ functions, finally to the formula for $XWORST$. Hence we demonstrated a potentially generalizable approach for deriving symbolic formulas for a function that takes symbolically stated functions among its arguments. Also of possible interest with respect to technique are distinctions among cases for which we obtained closed-form vs. non-closed form formulas. Note that our analytic objective of obtaining deterministic computations for prediction differs from that of Knuth [1975], who derived a predictor for individual problems (as opposed to ensembles of problems) that itself takes the form of a Monte Carlo experiment.

As mentioned in Section 6.1.1, the results of Section 3.4 extend what was previously known about how worst case cost varies with the weighting parameter W .

6.1.5 Experimental Tests of Predictive Power of the DEBET Model of A^*

Since the DEBET model (Chapter 3) is an abstract simplification of the experimental conditions of Chapter 2, it is desirable to assess how realistic its assumptions are by direct experimental tests of its predictive ability. Ours is the first

such experimental verification of the accuracy of an A* model in predicting the number of nodes expanded: no previous model included within its scope heuristic functions that occur in practice; no previous experiments amassed sufficient data against which to compare analytic predictions in a meaningful way.

We measured the accuracy of the DEBET model in predicting the worst case performance of A* using each of three 8-puzzle heuristic functions, for each of 11 values of W , over a set of 895 distinct problem instances of the 8-puzzle spanning a range of 25 values of N . In all, the body of observations against which the predictions are tested are derived from more than 26,000 distinct algorithm executions. Any future model of A* performance can test its predictions against these data.

We observed a difference between prediction and observation of 12% in the most accurate case observed, and disagreement by a factor of more than 190 in the most inaccurate case tested. For all 11 values of W tested, the predictions for heuristic function K_1 were accurate over the range of values of N to within a factor of 3 by the $E(K,W)$ measure (see Figure 3.5-9). For heuristic function K_2 , predictions for all 11 values of W were accurate to within a factor of 10 by the same measure. Only for heuristic K_3 is a generally larger discrepancy observed over the range of W . Against Knuth's "right order of magnitude" standards for predictive accuracy [Knuth 1975, p. 132], the DEBET model measures quite well for the majority of the cases tested, remarkably so in view of its stringent simplifying assumptions. Note further that the observed disagreement between prediction and observation is in itself revealing: apparently extreme worst case behavior does not occur in practice.

6.1.6 Abstractions in Analysis of Algorithms

We analyzed in Chapter 3 an algorithm schema parameterized by functions, as opposed to a schema parameterized by scalars as in other reports in the analysis of algorithms literature (e.g., Sedgewick's [1975] analysis of a version of the Quicksort algorithm that takes the median of k elements at a certain step). A scalar control policy parameter imposes a total ordering on a schema of algorithm instances, and it is natural to determine how cost or some other measure varies with the scalar parameter. In the DEBET model, we imposed a partial ordering on the schema of A*

algorithm instances, and determined how relative cost of two algorithm instances (i.e., two heuristic functions) relates to the position of the two instances under that partial ordering.

Specifically, we proved several monotonicity theorems over a lattice of algorithm instances (i.e., Theorems 3.2-5, 3.2-6, 3.2-7, 3.3-3, and 3.3-4). Here briefly is what they mean: In the DEBET model the distinct heuristic functions for A* are characterized by distinct pairs of functions $KMIN(i)$ and $KMAX(i)$ from the natural numbers to the non-negative reals. Each such $\langle KMIN, KMAX \rangle$ function pair can be considered a distinct algorithm instance. At the end of Section 3.3 we showed that a subset of all such $\langle KMIN, KMAX \rangle$, namely the set of "IM-never-overestimating" heuristic functions, can be partially ordered in a certain way and in fact satisfies the properties of an infinite continuous lattice. Each heuristic function in the IM-never-overestimating class can be expressed in terms of a relative error function $\delta(i)$ from the natural numbers to the real interval $[0,1]$. Corresponding to each such δ in this class is a value $XWORST'(\delta)$ representing the performance of that algorithm instance, and that "value" is itself a function of N and M (the depth and breadth respectively of the tree that is searched). We imposed an intuitively meaningful partial ordering on the set of all such performance functions. The analysis derives the mapping from each δ to its corresponding performance function. Theorem 3.3-3 shows that the mapping from the set of algorithms to the set of corresponding performance functions preserves the partial ordering. That is, if one algorithm instance is less than another under the partial ordering of the algorithm instance lattice, then likewise the performance function corresponding to the first algorithm is less than that corresponding to the second algorithm, under the partial ordering of the performance function lattice. Hence this monotonicity result is a powerful and succinct statement about the relation between arbitrary algorithm instances in this class. The other monotonicity theorems (Theorems 3.2-5, 3.2-6, 3.2-7, and 3.3-4) can be construed similarly as statements about lattices of algorithms.

Section 1.3 describes a manner of comparing different computational models for a given problem domain, i.e., of determining the accuracy of results in one computational model as predictions of analogous results in another computational model. Our experimental test of the predictive accuracy of the DEBET model (Section 3.5) constitutes just such a cross-model comparison. Likewise, the experimental comparison of the SAP-S and SAP-N- k_i -L models in Section 4.4.2 (i.e., N-Queens vs. Random-N-Queens) constitutes another example. In the former case we compared analytic results with experimental results; in the latter, experimental results in the SAP-S model with experimental results in the SAP-N- k_i -L model.

6.1.7 10^4 Experimental Observations for Future Theories to Predict and Explicate

Theories are necessarily about something. Most scientific theories attempt to make testable predictions. Chapters 2 and 4 present a large body of quantitative algorithm performance measurement data for future theories to try to predict and explicate.

The existence of this body of data constrains future conjectures in these domains: (1) hypotheses must be stated in enough detail to identify the existing data about which they speak, if any; (2) the conjectures must agree with the existing data to within statistical limits, or otherwise be immediately discarded or further qualified to exclude the contradictory data. For this purpose, Appendix C tabulates most of the experimental data plotted in the figures of this dissertation.

6.1.8 Cross-domain Comparisons

As mentioned in Chapter 1, both the A* and the SAP domains span a large number of variegated problem instances, and in both domains an important practical and theoretical objective is prediction of algorithm performance for an individual problem instance rather than over an ensemble of problem instances.

In both domains we compared two computational models by algorithm

performance: A* graph model with the DEBET model in the A* domain, and SAP-S with SAP-N- k_i -L in the SAP domain. Note however the difference in the means to obtain these results: A* graph and SAP-S results were obtained by experiment, whereas DEBET results, obtained by analysis, contrast with SAP-N- k_i -L results, obtained by experiment because analysis has yet to achieve the desired formulas in the SAP-N- k_i -L model.

In both domains, multiple performance measures reveal detail about algorithm behavior during a single search.

The absence of significant comparisons between the two domains, other than the above, indicates perhaps that the specific differences between these two domains outweigh their similarities.

6.2 Immediate Extensions

The man [sic] with a new invention is a crank until it works.

Mark Twain

It is clear that we have investigated but a small portion of a large, and as yet sparsely explored, research domain. The results obtained provide evidence about some extant questions, but also raise many new questions. It is clear throughout the text that numerous possible extensions to results of one sort were traded-off in favor of obtaining some results of another sort (see Section 1.5). Sections 2.9, 3.6, and 4.6, and Appendix B list a number of specific objectives for future work as direct extensions of the dissertation results. We shall not recapitulate here the technical detail of those sections. Instead, in this section we offer informal comments regarding the general directions motivating those potential extensions.

6.2.1 Mathematical Analysis of Algorithms for Satisficing Assignment Problems

The performance results given in Chapter 4 for algorithms for satisficing assignment problems are entirely experimental in nature. The definitions and traces of

these algorithms suggest certain general conjectures, which one may attempt to prove or disprove, including the following.

1. Prove that BACKMARK and BACKJUMP are valid algorithms for all SAPs, for SAPs defined in Definition 4-1, and using the definition of "valid" given in Section 4.5.3.
2. Prove for all SAPs that BACKMARK and BACKJUMP execute subsets of the pair-tests executed by BACKTRACK, and that BACKMARK executes exactly the same distinct pair-tests as BACKTRACK.
3. Derive formulas, for any SAP s , for $T_f(S)$, $D_f(S)$, and $M_f(S)$ for each of BACKTRACK, BACKMARK, BACKJUMP, and DEEB. It may be problematic to derive simple symbolic formulas for these quantities. An alternative is to restrict attention to the SAP- $N-k_1$ - L model, attempting to derive an average case or worst case formula for $T_f(N, k_1, k_2, \dots, k_N, L)$ for each of the four algorithms. (Even this may be problematic in the general case.)
4. Having derived formulas described in item 3, determine the subsets of SAPs for which: (a) BACKMARK has as small or smaller $T_f(S)$ than BACKJUMP and DEEB; (b) BACKJUMP yields smallest $T_f(S)$ of the three candidates; (c) DEEB yields smallest $T_f(S)$. Call these subsets $B_{\text{BACKTRACK}}$, B_{BACKJUMP} , and B_{DEEB} , respectively. Determine a test for membership in each of the three subsets.

The present random- N -Queens and ISVL results (Sections 4.4.2 and 4.4.3) are estimates of the values of $T_f(N, k_1, \dots, k_N, L)$ for particular cases. Comparison of the Random- N -Queens values with the corresponding values for N -Queens SAPs (Figure 4.4.2-1) show clearly the limited ability of the SAP- $N-k_1$ - L model to predict the cases tested.

6.2.2 Analogous Experiments with Different Problems

The present experiments span only a few sample sets of problems (one for A^* , four for SAP search). The resulting data do not permit predictions about the performances of the same algorithms when applied to problems other than those tested here. Hence generalizations based on experiment about the performances of these algorithms must await the completion of methodologically comparable experiments for other problems.

One issue arising in designing such experiments is the choice of problems for which to measure algorithm performance. Besides the tradeoffs mentioned in Section 1.5, one can choose problems similar to those already studied (a "concentrated" or

"variations on a theme" approach), or problems very different from those already studied (a "shotgun" approach).

One extreme is to choose for new experiments a problem that is identical in all but one parameter (e.g., size, degree of constraint, number of solutions, and so on) to a problem already investigated. Examples of such extensions in parameter range include the 100-Queens problem, the 100-random-queens problem, the ISVL problems for other values of L , and the 15-puzzle. Advantages of this approach are that it permits relatively controlled comparisons (if one problem specification parameter varies while the others are held fixed), and that the observed differences may exhibit sufficient regularity as to suggest the possibility of accounting for the differences analytically. An obvious disadvantage is that results for the particular cases tested may not generalize. To this impediment the investigator may respond (laboriously) by supplying data covering additional cases, obtained by varying the combinations of the experiment parameters.

The results of a "shotgun" approach may prove useful in making taxonomic classifications. Noteworthy phenomena observed of one problem may not be exhibited by another dissimilar problem, hence such results may permit drawing gross limits to the generality of an observed phenomena. Two examples suffice here to make the point concrete. In Chapter 2 we observed the existence of the "crossover N decreases with increasing N " phenomena (see Section 6.1.1). Admitting that this phenomena may not extend to all possible problem graphs, to which then does it apply? Section 6.1.1 mentions, for example, concerning the relative performances of backtracking and constraint satisfaction for problems having complete consistency graphs vs. those having incomplete consistency graphs.

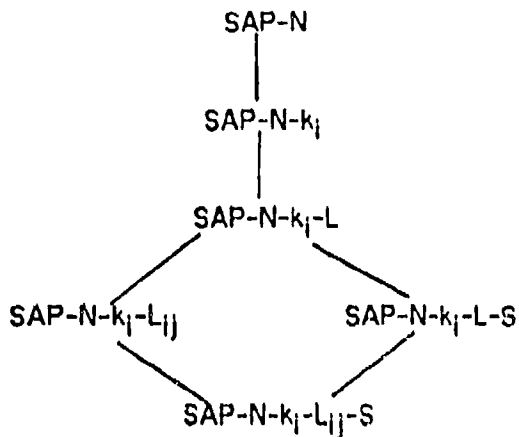
Lacking definitions for direct measures of the degree of similarity between two problems, the present approach is to base statements about "problem similarity" in terms of quantitative measurements of the performances of algorithms that solve the problems, i.e., two problems are similar to the extent that the performances of an algorithm in solving them are similar. Such evidence may be useful in guiding the development of theories about problem similarity.

6.2.3 "Successive Set Partitioning": More Parameters

We suggest here how the successive set partitioning technique defined in Section 4.4.1 can be extended to permit a finer grain comparison of N-Queens SAPs with the parametrically similar "random-N-Queens" SAPs. The experiments in Section 4.4.2 fail to account for the fact that the randomly generated SAPs, while identical in size and degree of constraint to the corresponding N-Queens SAP, in general do not have the same total number of solutions as the N-queens SAP. Hence it would be interesting to replicate those experiments, but including among the randomly generated problems only those having the same total number of solutions as the corresponding N-Queens SAP. In formal terms, partition each $N-k_i-L$ equivalence class into equivalence classes such that members of a given equivalence class have the same number of solutions. Hence this defines a new computational model for SAPs, which we shall call the SAP- $N-k_i-L-S$ model. Then the experiments measuring algorithm performance for Random-N-Queens problems can be replicated for various values of S , including the value of S corresponding to an actual N-Queens problem. Of course, generating SAPs in a given $N-k_i-L-S$ equivalence class may be more difficult than generating those in a $N-k_i-L$ equivalence class.

An alternative to the $N-k_i-L-S$ refinement of the $N-k_i-L$ model is suggested by the notion that there is more to constraint in a SAP than is expressed in a single scalar value L . In what we call the $N-k_i-L_{ij}$ model the single scalar L problem specification parameter is replaced by a parameter L_{ij} for each i and j . We define $L_{ij} = |P_{ij}| / k_i * k_j$. (Hence L equals the sum over i and j of L_{ij} , divided by D_{max} .) This formulation reflects the possibility that the observed differences between N-queens and random-N-queens SAPs is due in part to the differences in the corresponding L_{ij} values, even though the overall L value is the same. In contrast with the SAP- $N-k_i-S$ model, in which it is computationally expensive to determine the value of S for a particular SAP, the values of L_{ij} for a given SAP are about as easy to determine computationally as the value of L , the gross measure.

Since the L_{ij} and the S partitions are independent, they may be combined into a single $N-k_i-L_{ij}-S$ model. The diagram below depicts, in the form of a lattice, these various successive partitions of the set of all SAPs satisfying Definition 4.1.



The range of values of an algorithm performance measure (e.g., T_f) over the members of a given equivalence class provides a quantitative measure of the grain of the partition. For example, if the maximum and minimum values of T_f (for BACKTRACK, say) over the members of one equivalence class differ by a factor of two, then *ipso facto* the average value of T_f for this equivalence class will predict the value of T_f for any individual member of the class to within a factor of two. Hence given a sufficiently fine grained partition, analytic or experimental average performance results for an ensemble of problem instances (i.e., an entire equivalence class) can be good predictors of performance of an individual problem instance (i.e., an individual member of the equivalence class). An objective then is to define problem specification parameters yielding such a fine-grained partition of the problem class.

6.2.4 Further Analysis of the DEBET Model of A*

We divide our immediate future efforts concerning the DEBET model into four categories: 1) prove more theorems; 2) reformulate present theorems in terms of an infinite continuous lattice of heuristic functions; 3) evaluate the formula derived in Theorem 2 numerically over a systematically chosen range of argument values; 4) test the accuracy of predictions using the DEBET model of A* for additional problems, based on KMIN(i) and KMAX(i) measurements and XMAX(K,W,N) measurements.

Regarding the first category, we note that the theorems in Chapter 3 fall into

two categories: those spanning all heuristic functions of the type defined there (i.e., Theorems 3.1-1 and 3.1-2), and those theorems spanning only a subset thereof (i.e., the remaining theorems). In particular we seek simpler formulas for $XWORST(M,N,KMIN,KMAX,W)$ than those reported here (particularly simple closed formulas), and also more theorems (such as monotonicity theorems) relating various combinations of $KMIN$, $KMAX$, and W . In particular, we wish to prove several theorems concerning the dependence of $XWORST$ on W .

Regarding the second category, we seek to determine whether a formulation in terms of a lattice of functions simplifies the present proofs (e.g., due to notational abstractions), and also to seek new theorems such as described under category 1.

Regarding the third category, we note that although we obtained a simple symbolic formula valid for a broadly defined set of heuristic functions (Theorem 3.3-2), in the most general case we obtained non-closed form formulas whose values can be computed quickly by a simple algorithm. Preliminary results not reported here make it seem likely that closed form results will be even more difficult to obtain for average case analysis of A^* than for the present worst case analysis. Hence we have a reasonable expectation that additional non-closed-form results will appear in these domains. We may evaluate such formulas over various particular cases in an attempt to discover patterns about which we may attempt to prove theorems as in category 1. We might attempt in fact to automate this task to the extent we can specify what sort of patterns to seek.

Regarding the fourth category, we note that the scope of Theorems 3.1-1 and 3.1-2 permit predictions to be made for arbitrary problems and heuristic functions of the sort occurring in practice, so that each problem for which experimental data is obtained, of the sort reported in Chapter 2, permits an additional test of the DEBET model. Tests spanning a number of problems besides the 8-puzzle may clarify the predictive properties of the DEBET model in more detail than this first test permits.

6.2.5 Methodological and Practical Issues for Experiments

The experiments reported in the dissertation consumed in total about 50 hours

of cpu-time on a DEC KL-10. The fact that the present experiments revealed numerous new phenomena that previous investigators with more limited available computing power could not have discovered indicate long-time-scale algorithm performance measurement experiments as a potentially productive area for obtaining interesting and precise results as a guide to theoretical development. We have proposed extensions to the dissertation of this sort, and others similar are easy to conceive. The experimental methodology we have used is reasonably straightforward, and lends itself to adaptation in other cases. Hence it would seem that a lot of well-specified future experimental work awaits attention. (See Appendix B.)

As a possible reference for investigators who may pursue such ends, it is appropriate that we report some of our experience with the practical matters of conducting such experiments. A critical issue that must be faced in doing experiments of this sort is that of being overwhelmed by the sheer volume of data produceable by long-time-scale experiments. To measure fewer distinct quantities than is possible (e.g., only T in Chapter 4, as opposed to T , D , and M) is to run the risk of missing interesting phenomena (especially concerning the relations between the several measures; see also Section 6.3.3). Our experience suggests the desirability of more sophisticated automated procedures for data bookkeeping and analysis than were implemented in this study. Even so routine a matter as plotting figures of experimental data can have a considerable impact on possible results, since in a situation where many such plots are possible, one must choose which ones actually to produce. The fact that a table of numbers may fail to suggest what is graphically evident when the same numbers are plotted suggests the desirability of integrating the plotting task into a single experimental system that accepts specifications of experiments interactively, executes them, collects, analyzes and processes the data into a variety of output forms, and produces a machine-readable permanent record of the results.

Our experience suggests the desirability of building generality into the experiment specification mechanism. As described in Chapter 5, our apparatus are capable of performing a combinatorially large number of experiments, simply by specifying interactively the choice of algorithm, size and number of problem instances, solution criteria, heuristics, and other control policy and problem specification parameters. In particular, several of the experiments proposed as extensions to the

dissertation can at this time be executed by the present apparatus, requiring no more than about a minute of interactive time to specify the parameters. Similarly, to apply the present apparatus to other problems requires only the definition of a few problem-specific external procedures.

Note incidentally that one possible practical application of improved predictability of algorithm performance is to estimate the amount of cpu time required to execute a proposed experiment.

Another methodological issue arising in the design of experiments that measure aggregate performance statistics over sets of randomly selected problem instances is that of defining the set of all problem instances having a certain combination of problem specification parameter values, and that of insuring that the selection mechanism is independent and uniform. For example, in the domain of map coloring, what constitutes the set of all maps having N regions?

6.2.6 Other Issues Excluded from the Dissertation

A number of important issues completely excluded from consideration as beyond the present scope of the present work are listed below.

Cost to compute the value of an A* heuristic function

For practical reasons, obviously it is important to know not only how many nodes are expanded by A* when conducting a search using a particular heuristic function to order the node expansions, but also how much computational effort is required to compute the value of the heuristic function at each node.¹ In contrast, in the experiments of Chapter 2 we simply counted the number of nodes (or other quantities). It would be interesting to relate these measures, for each of several heuristic functions, to the corresponding cpu-time required to execute the heuristic function at each step. Is a more accurate but harder to compute heuristic function

¹ For example, a heuristic function can compute exactly the distance between arbitrary nodes simply by doing a breadth-first search from one node to the other within its "black box"; in practice, this implementation of the "perfect" heuristic would be undesirable.

cost effective?

Representation of A* heuristic functions

Similarly, practical concerns may motivate an interest in how compactly the algorithm for computing the value of a heuristic function for A* can be represented: as a simple formula, as a large procedure, as a table of values obtained by preprocessing, and so on. However, in our computational model such questions cannot be addressed formally, because a heuristic function is treated simply as a mathematical function. Hence it would be interesting to investigate time-space tradeoffs, for example relating the number of nodes expanded by a heuristic function to the memory size required to implement the heuristic function, for each of several heuristic functions.

Cost-effectiveness of analytic predictions for A*

As discussed in Section 1.3.2, our ability to compare analytic predictions of A* performance with experimental observations depends on the ability to identify the actual parameters to the XWORST function corresponding to the experimental conditions. Since these values of KMIN(i), KMAX(i), and M are determined by experiment, our procedure for predicting performance may not be practical in its current form, except possibly as a basis for choosing a value of W. This is entirely consistent with our limited current objective: to demonstrate the technical feasibility of such predictions, leaving to future work refinements that may improve practical utility.

Symmetry and other representation issues in SAP search

It has been widely observed that the apparent difficulty of solving a given problem may depend upon the way in which it is represented. The mutilated checkerboard problem and number scrabble are familiar examples of problems whose solution is greatly facilitated by certain reformulations of the statement of the problem. When such problems can be modeled as state space problems, the apparent effect of such a reformulation is a reduction in the size of the space that must be searched.

Instances of this representation issue arise in the case of assignment problems

considered in Chapter 4 -- there are many different formulations of the eight queens problem that satisfy the definition of a SAP given in Chapter 4. In one formulation, there are eight problem variables, corresponding to the queens, and each queen can be assigned to any of the 64 squares of the board. Realizing that no row of the board can contain more than one queen in a solution suggests a formulation in which the queens are restricted to distinct rows, hence reducing the number of possible assignments of each queen from 64 to eight. Left-right symmetry of a chess board suggests a further refinement in which one arbitrarily chosen queen is restricted to the leftmost four squares of its row (this is the formulation used in Chapter 4). Additional symmetry considerations may suggest further refinements. Alternatively, one might formulate the 8 queens problem as having 64 variables, each of which may take on the values 1 or 0, representing respectively the presence or absence of a queen.

The performance of a given algorithm for SAP search, such as the backtrack algorithm, varies with the formulation of the problem to be solved. Hence it would be interesting to compare algorithm performance among several such formulations, providing a basis for comparing the formulations by the criteria of algorithm performance.

6.3 Long-Term Objectives

The present results also suggest a number of other objectives for future work broader in scope than those listed in Section 6.2, and more long term in time scale.

6.3.1 Error Analysis in Cross-model Comparisons

Section 3.5 identifies a number of factors contributing to the observed disagreement between the predictions of the DEBET model, as applied to the 8-puzzle heuristics, and the observed performance measurements for the 8-puzzle. It would be interesting to determine how much of the discrepancy can be attributed individually to each of these factors.

6.3.2 Algorithm Behavior: What is Observable? What is Controllable?

As discussed in Section 6.1.1 and elsewhere, we have defined certain performance measures as combinations of other measures (e.g., in the SAP domain $T=D+M$), with the effect of revealing in greater detail the characteristics of algorithm behavior during a single search. Extending this approach depends on the ability to define new performance measures. To make the point concrete, we suggest now several in the SAP domain: The quantity $D_t/D_{\max}$ or $D_a/D_{\max}$ measures the fraction of all distinct possible pair-tests that are executed during a single search. As a second example, let $D = F + G$, where F = number of pair-tests executed exactly once during a single search, and G = number of pair-tests executed more than once. Hence $M = 1$ iff $G = 0$. The pair-tests counted by G are those to focus on in attempts to devise an algorithm for SAPs that is optimal by the M measure.

Clearly then, in these domains we can measure algorithm behavior to the extent we can specify exactly what to measure. The point intended here is that we have not yet exhausted all possible definitions of what to measure. The more detailed the performance measurements we obtain, the more detailed behavior we have the potential of observing. The present results contribute to a steadily growing body of evidence that even relatively simple algorithms, such as A^* and backtracking, when applied to relatively simple problems, such as the Eight Puzzle and the Eight Queens

problem, can exhibit very complex behavior or a very broad range of behavior. What we can observe and measure we can attempt to control, by devising new variations of an algorithm, as we have demonstrated by the invention of BACKMARK and BACKJUMP. Hence we observe that new performance measures sometimes suggest new algorithms.

Devising new control policy parameters for an algorithm can introduce new opportunities for controlling its behavior. One example studied here is the dependence of search performance on the candidate value ordering in solving satisficing assignment problems. In this case we can freely choose in practice any one of the permissible permutations of candidate values, and we demonstrated that choosing randomly can in some cases give better performance than using some "obvious" ordering. Another example in the present work is the DEELEV algorithm, in which we exploited the fact that one can use some form of backtracking to find a partial solution of specified size before commencing the execution of a Waltz-type constraint satisfaction algorithm. In this case as well the introduction of an additional control policy parameter (l, the level at which to switch algorithms) afforded the opportunity to improve performance.

There is nothing new about our application of these general approaches of introducing additional performance measures and control policy parameters, and nothing that depends on peculiarities of the present case studies. Indeed their application is implicit in many investigations of algorithms. Our point is simply to draw attention to their potential applicability in diverse circumstances.

6.3.3 The 8-puzzle as a Highly Regular Graph

The 8-puzzle is treated in this dissertation mathematically as a graph, but what does that particular graph look like? Figure 2.1-1 depicts a small portion of that graph, showing that portion to appear as what might be called a "near-tree". Indeed, the heuristic functions we have considered for the 8-puzzle depend for their effectiveness on properties that hold for that particular graph. Heuristics that reduce search cost for the 8-puzzle in general will be ineffective or inapplicable to other graph search problems. Hence to analyze a heuristic function and account for its performance in detail we must, it would seem, determine what regularities of the graph the heuristic function exploits.

The following example will illustrate the possibility of a sort of "topographic" analysis of heuristics. Consider the $\text{seq}(s)$ term in the definition of heuristic function K_3 . This term has the value 0 iff the order of the perimeter tiles in the present configuration s matches that of the perimeter tiles in the goal configuration up to rotation. Assuming the hole is on the perimeter, 7 tile moves are required to rotate the 7 perimeter tiles by one position (either clockwise or counter clockwise). Iterating this move sequence 8 times brings the tiles back to their original configuration, s . Hence the locus of vertices in the 8-puzzle graph for which $\text{seq}(s)=0$ is a loop of length 56. Since the value of $\text{seq}(s)$ is greater than zero for vertices not on this loop, it is easy to picture in a somewhat vague way the $\text{seq}(s)$ term in the form of a "topographic" map, in which the value of $\text{seq}(s)$ is the "elevation" above the "plane" of the graph. In this picture, we see that the locus of $\text{seq}(s)=0$ is a closed "trench", surrounded inside and outside by higher "elevations". Noting that the goal node is either in the trench or within a few moves of it, we see a possible "explanation" for the superb efficiency of the K_3 heuristic function: since best-first search follows paths to lower elevations, the effect is to find a path from the root node to the bottom of the trench and then follow it to the goal. (For details, consider the relative values of the terms in the definition $K_3(s) = K_2(s) + 3*\text{seq}(s)$.) This " K_3 follows a trench" hypothesis can be subjected to a simple experimental verification, although we have not yet done so. Presumably such an experiment may reveal certain details omitted above.

A heuristic function embodies some partial information about the structure of a problem graph. Clearly, we understand little as yet about the nature of "problem structure" and its encoding into "heuristic knowledge", i.e., into a function.

6.3.4 Toward Theories About "Problem Structure" and "Heuristic Knowledge"

We make no claim that the present results about the "knowledge" encoded in a heuristic function or about "problem structure" capture anything but a fragment of the characteristics we associate intuitively with these elusive concepts. The limited results we have obtained at least are mathematically precise and experimentally verifiable. Our

approach in each case has been to speak of certain observable manifestations of the concept, i.e., to speak in terms of the effect on algorithm performance rather than attempting to define a measure of structure or knowledge directly. The greater challenge, it would seem, is to devise theories that define the concepts directly, in such a way that theoretical results can be tested against observation, for example against the observations reported in this dissertation.

Various aspects of problem structure or complexity have been investigated. Michie [1968] takes an information theoretic approach to what makes a problem hard. Amarel [1968] and Cohen [1977] uncover symmetries in a problem graph. Gaschnig [1979] describes an approach, based upon a notion of problem similarity, that can be used when attempting to devise a heuristic for a given problem graph. The latter approach relies on a change in perspective: instead of seeking a heuristic directly for a given problem Q1, one seeks instead a problem Q2 that is easier to solve than Q1 and related to Q1 in a certain way. This "transfer problem" is then applied in a certain way as a heuristic function for Q1.

6.3.5 Performance Analysis in AI Research: General Comments

The experiments and analysis reported in this thesis were undertaken with the hope of contributing specific results, but also to explore further the use of performance measurement and analysis methodologies in studies of Artificial Intelligence. Of course, the present specific results speak only about the particular domains studied, but in so doing they add further support to general statements, metaconclusions if you like, that may be relevant to AI research more generally:

1. Relatively simple algorithms applied to relatively simple problems can produce a broad and complex range of behavior.
2. Measuring the performance of a problem solving system can reveal details about its behavior that promote useful insight and may suggest improvements.
 - 2a. The more cases (input problems) tested, the more likely that the results show the current understanding to be simplistic (e.g., not valid for all cases and hence in need of qualification) or otherwise inadequate; the more likely also that the results reveal the existence of previously unsuspected phenomena.

- 2b. The more distinct performance measures collected, the better one can identify sources of inefficiency or error, and respond accordingly.
- 2c. The more variations of an algorithm tested, the easier it is to identify the dependence of performance on the various control policy options, and hence to choose appropriate options.
3. Open questions formulated in precise terms, and having definite and reproduceable answers, promote progress in ways that vaguely stated questions usually do not.

Historically, for a variety of reasons, much more effort has usually been devoted to designing and constructing AI problem solving systems than to measuring their resulting performance. Many productive performance evaluations have been performed, not only for relatively simple search algorithms but also concerning game playing (e.g., [Berliner 1974], [Gillooly 1978], [Findler 1978], [Sugar 1975], [Samuel 1963, 1967]), speech understanding (e.g., [Paxton 1977], [Goldberg 1975], [Smith 1977]), medical diagnosis (e.g., [Yu, et al. 1978a], [Yu, et al. 1978b]), and other domains. Nevertheless, opportunities abound for extensive objective tests of many AI problem solving systems, both of recent and of older construction. In contrast, it is highly unlikely that physicists who have completed the construction of a new more powerful particle accelerator would unplug it, to turn to other matters, as soon as it passed the acceptance tests. We think it clear that AI researchers can profit by the physicists' example, by considering AI problem solving systems as scientific instruments, sources of hard data by which to define and investigate problem solving behavior in detail, as well as ends in themselves.

References

- Aho, A., J. Hopcroft and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Mass. 1974.
- Amarel, S., "On Representations of Problems of Reasoning about Actions," in *Machine Intelligence 3*, D. Michie (ed.), American Elsevier Publ. Co., New York, 1968.
- Barlow, R.E., Bartholomew, D.J., Bremner, J.M., and Brunk, H.D., *Statistical Inference Under Order Restrictions*, John Wiley & Sons, Inc., New York 1972.
- Barstow, D., "A Knowledge-Based System for Automatic Program Construction," *Proc. 5th Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, Cambridge, Mass., August 1977.
- Barstow, D., and E. Kant, "Observations on the Interaction of Coding and Efficiency Knowledge in the PSI Program Synthesis System," *Proc. 2nd Intl. Conf. Software Engineering*, San Francisco, October 1976, pp. 19-31.
- Berliner, H. J., "Chess as Problem Solving: The Development of a Tactics Analyzer," Ph.D. Thesis, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., March 1974.
- Berliner, H., "Experiences in Evaluation with BKG -- A Program that Plays Backgammon," *Proc. Intl. Joint Conf. on Artificial Intelligence*, Cambridge, Mass., August 1977.
- Berliner, H.J., "The B* Tree Search Algorithm: A Best-first Proof Procedure," Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pa., April 1978.
- Birkhoff, G., *Lattice Theory* (prelim. 3rd ed.), American Mathematical Society, 1963.
- Bobrow, D., and B. Raphael, "New Programming Languages for AI Research," *Computing Surveys*, Vol. 6, 1974, pp. 153-174.
- Brown, T., "A Note on 'Instant Insanity'," *Mathematics Magazine* (41), No. 4 (1968), pp. 167-169.
- Busacker, R., and T. Saaty, *Finite Graphs and Networks*, McGraw-Hill Book Co., New York 1965.
- Chang, C., and J. Slagle, "An Admissible and Optimal Algorithm for Searching And/Or Graphs," *Artificial Intelligence*, Vol. 2, 1971.
- Cohen, B., "The Mechanical Discovery of Certain Problem Symmetries," *Artificial Intelligence*, Vol. 8, No. 1, 1977, pp. 119-131.
- David, H.A., *Order Statistics*, John Wiley & Sons, Inc., New York 1970.

- DeGroot, M., Probability and Statistics, Addison-Wesley Publ. Co., Reading Mass., 1975.
- Dijkstra, E., "A Note on Two Problems in Connexion with Graphs," *Numerische Mathematik* Vol. 1, 1959, pp. 269-271.
- Doran, J., "An Approach to Automatic Problem Solving," in *Machine Intelligence* 1, N. Collins and D. Michie (eds.), American Elsevier Publ. Co., New York 1967.
- Doran, J., "New Developments of the Graph Traverser," in *Machine Intelligence* 2, E. Dale and D. Michie (eds.), American Elsevier Publ. Co., New York 1968.
- Doran, J., and D. Michie, "Experiments with the Graph Traverser Program," *Proc. Royal Society of London, Series A*, Vol. 294, 1966, pp. 235-259.
- Drake, A., *Fundamentals of Applied Probability Theory*, McGraw-Hill Book Co., New York 1967.
- Eastman, C., "Preliminary Report on a System for General Space Planning," *CACM*, Vol. 15, No. 2, February 1972.
- Eastman, C., "Automated Space Planning," *Artificial Intelligence*, Vol. 1, 1970 pp. 27-120.
- Felgenbaum, E.A., "The Art of Artificial Intelligence: Themes and Case Studies of Knowledge Engineering," *Proc. Intl. Joint Conf. on Artificial Intelligence*, Cambridge, Mass., 1977, pp.1014-1029.
- Feller, W., *An Introduction to Probability Theory and its Applications*, vol. 1, John Wiley & Sons, Inc., New York 1968.
- Fikes, R. E., "REF-ARF: A System for Solving Problems Stated as Procedures," *Artificial Intelligence*, Vol. 1, 1970, pp. 27-120.
- Fillmore, J., and S. Williamson, "On Backtracking: A Combinatorial Description of the Algorithm," *SIAM J. Computing*, Vol. 3, No. 1, March 1974, pp. 41-55.
- Findler, N. V., "Computer Poker," *Scientific American*, July 1978.
- Floyd, R. W., "Nondeterministic Algorithms," *J. Assoc. Computing Machinery*, Vol. 14, 1967, pp. 636-644.
- Freuder, E., "Synthesizing Constraint Expressions," *Comm. ACM*, Vol. 21, No. 11, November 1978. Also appeared as M.I.T. AI memo 370, July 1976.
- Fuller, S., J. Gaschnig and J. Gillogly, "Analysis of the Alpha-Beta Pruning Algorithm," *Carnegie-Mellon University, Dept. of Computer Science*, Pittsburgh, PA, July 1973.
- Gardner, M., *Mathematical Games*, *Scientific American*, Vol. 227, 1972, pp. 176-182.

- Garey, M., and Johnson, D., "Approximation Algorithms for Combinatorial Problems: An Annotated Bibliography," in Algorithms and Complexity, J.F. Traub (ed.), Academic Press, New York 1976.
- Gaschnig, J., "A Constraint Satisfaction Method for Inference Making," Proc. 12th Annual Allerton Conf. Circuit and System Theory, U. Ill. Urbana-Champaign, Oct. 2-4, 1974.
- Gaschnig, J.G., "Design and Construction of a General Purpose State Space Searching Apparatus," unpublished memo, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., December 1975.
- Gaschnig, J., "Exactly How Good are Heuristics?: Toward a Realistic Predictive Theory of Best-First Search," Proc. 5th Intl. Joint Conf. on Artificial Intelligence, Cambridge, Mass., August 1977 (1977a).
- Gaschnig, J., "A General Backtrack Algorithm that Eliminates Most Redundant Tests," Proc. Intl. Joint Conf. Artificial Intelligence, Cambridge, MA, August 1977 (1977b).
- Gaschnig, J., "Experimental Case Studies of Backtrack vs. Waltz-type vs. New Algorithms for Satisficing Assignment Problems," Proc. 2nd Conf. Canadian Society for Computational Study of Intelligence, Toronto, Ont., July 1978.
- Gaschnig, J., "A Problem Similarity Approach to Devising Heuristics," to appear in Proc. 6th Intl. Joint Conf. on Artificial Intelligence, Tokyo, August 1979.
- Gelperin, D., "On the Optimality of A^* ," Artificial Intelligence, Vol. 8, 1977, pp. 69-76.
- Gillogly, J. J., "Performance Analysis of the Technology Chess Program," Report CMU-CS-78-109, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pennsylvania, March 1978.
- Golomb, S. and L. Baumert, "Backtrack Programming," J.A.C.M., Vol. 12, No. 4, October 1965, pp. 516-524.
- Goldberg, H. G., "Segmentation and Labeling of Speech: A Comparative Performance Evaluation," Technical Report, Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pa., December 1975.
- Green, C., "A Summary of the PSI Program Synthesis System," Proc. 5th Intl. Joint Conf. on Artificial Intelligence, Cambridge, Mass., August 1977.
- Gumbel, E. J., Statistics of Extremes, Columbia University Press, New York, 1958.
- Haralick, R. and L. Shapiro, "The Consistent Labeling Problem: Part I," IEEE Transactions on Pattern Analysis & Machine Intelligence, Vol. 1, No. 2, April 1979 (1979a).
- Haralick, R. and L. Shapiro, "The Consistent Labeling Problem: Part II," IEEE Transactions on Pattern Analysis & Machine Intelligence, to appear in 1979 (1979b).

- Harris, L., "Heuristic Search Under Conditions of Error," *Artificial Intelligence*, Vol. 5, No. 3, 1974, pp. 217-234.
- Hart, P., N. Nilsson and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Trans. Sys. Sci. Cybernetics*, Vol. 4, No. 2, 1968.
- Hayes, J., et. al., "A Quantitative Study of Problem-Solving Using Sliding Block Puzzles: the 'Eight Puzzle' and a Modified Version of the Alexander Passalong Test," Experimental Programming report no. 7, Experimental Programming Unit, University of Edinburgh 1965.
- Hayes-Roth, F., and V. Lesser, "Focus of Attention in the Hearsay-II Speech Understanding System," *Proc. 5th Intl. Joint Conf. on Artificial Intelligence*, Cambridge, Mass., August 1977, pp. 27-35.
- Ibaraki, T., "Theoretical Comparisons of Search Strategies in Branch-and-Bound Algorithms," *International Journal of Computer and Information Sciences*, Vol. 5, No. 4, December 1976, pp. 315-344.
- Jackson, P. *Introduction to Artificial Intelligence*, Petrocelli Books, New York 1974.
- Johnson, D.S., "Approximation Algorithms for Combinatorial Problems," *Proc. 5th Annual ACM Symposium on Theory of Computing*, 1973, pp. 38-49; also in *Journal of Computer and System Sciences* (9), no. 3, Dec. 1974, 256-278.
- Kadane, J., Private communication, January 1978.
- Kant, E., "The Selection of Efficient Implementations for a High-Level Language," *Proc. Symposium on Artificial Intelligence and Programming Languages*, University of Rochester, August 1977. Appears in joint issue of *SIGPLAN Notices*, vol. 12, no. 8, August 1977, and *SIGART Newsletter* No. 64, August 1977.
- Karp, R., "The Probabilistic Analysis of Some Combinatorial Search Algorithms," In *Algorithms and Complexity*, J.F. Traub (ed.), Academic Press, New York 1976.
- Knuth, D. E., *The Art of Computer Programming; Volume 2. Semi-Numerical Algorithms*, Addison-Wesley, Reading, Mass. 1969.
- Knuth, D. E., *The Art of Computer Programming; Volume 1. Fundamental Algorithms* (2nd Ed.), Addison-Wesley, Reading, Mass. 1973 (1973b).
- Knuth, D. E., *The Art of Computer Programming; Volume 3. Sorting and Searching*, Addison-Wesley, Reading, Mass. 1973 (1973b).
- Knuth, D.E., "Estimating the Efficiency of Backtrack Programs," *Mathematics of Computation* (29), no. 129, January 1975, pp. 121-136.
- Knuth, D.E., and Moore, R.W., "An Analysis of Alpha-Beta Pruning," *Artificial Intelligence* 6, 1975, pp. 293-326.

- Lauriere, J. L., "A Language and a Program for Stating and Solving Combinatorial Problems," *Artificial Intelligence*, Vol. 10, No. 1, 1978, pp. 29-127.
- Lindstrom, G., "Backtracking in a Generalized Control Setting," Report UUCS 77-105, Computer Science Dept., University of Utah, July 1977 (revised January 1979). To appear in *Trans. Programming Languages and Systems*, Vol. 1, No. 1.
- Lindstrom, G., "Efficiency in Nondeterministic Control Through Non-Forgetful Backtracking," Report UUCS-77-114, Computer Science Dept., University of Utah, October 1978. To appear in *Journal of Computer Languages*.
- Mackworth, A.K., "Consistency in Networks of Relations," *Artificial Intelligence*, Vol. 8, No. 1, February 1977, pp. 99-118.
- Martelli, A., "On the Complexity of Admissible Search Algorithms," *Artificial Intelligence*, Vol. 8, 1977, pp. 1-13.
- Medress, M., et. al., "Speech Understanding Systems: Report of a Steering Committee," *SIGART Newsletter*, no. 62, April 1977, pp. 4-8.
- Michie, D., "Strategy Building with the Graph Traverser," in *Machine Intelligence 1*, N. Collins and D. Michie (eds.), American Elsevier Publ. Co., New York 1967.
- Michie, D., "A Theory of Advice," *Machine Intelligence 8*, E. Elcock and D. Michie (eds.), Ellis Hartwood, Ltd., Chichester, England, 1977.
- Michie, D., and R. Ross, "Experiments with the Adaptive Graph Traverser," in *Machine Intelligence 5*, B. Meltzer and D. Michie (eds.), American Elsevier, New York 1970.
- Montanari, U., "Networks of Constraints: Fundamental Properties and Applications to Picture Processing," *Information Science*, Vol. 7, 1974, pp. 95-132.
- Munyer, J., "Some Results on the Complexity of Heuristic Search in Graphs," Technical report HP-76-2, Information Sciences Dept., U. Cal. Santa Cruz, September 1976.
- Munyer, J., and I. Pohl, "Adversary Arguments for the Analysis of Heuristic Search in General Graphs," Technical report HP-76-1, Information Sciences Dept., U. Cal. Santa Cruz, July 1976.
- Newell, A., J. Shaw and H.A. Simon, "Empirical Explorations with the LOGIC THEORY Machine: A Case Study in Heuristics," in *Computers and Thought* (E. Feigenbaum and J. Feldman, eds.), McGraw-Hill Book Co., New York 1963.
- Newell, A., and Simon, H. A., *Human Problem Solving*, Prentice-Hall, Inc., Englewood Cliffs, N. J. 1972.
- Nijenhuis, A., and Wilf, H.S., *Combinatorial Algorithms*, Academic Press, New York 1975.
- Nilsson, N., "Artificial Intelligence," *Proc. Information Processing (IFIP) 74*, North-Holland Publ. Co. 1974.

- Nilsson, N., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill Book Co., New York 1971.
- Paxton, W. H., "A Framework for Speech Understanding," Technical Note 142, Artificial Intelligence Center, SRI International, Menlo Park, California (June 1977).
- Pohl, I., "First Results on the Effect of Error in Heuristic Search," Machine Intelligence 5, B. Meltzer and D. Michie (eds.), Edinburgh University Press, Edinburgh 1970 (1970a).
- Pohl, I., "Heuristic Search Viewed as Path-Finding in a Graph," Artificial Intelligence, Vol. 1, 1970 (1970b).
- Pohl, I., "Bi-Directional Search," in Machine Intelligence, Vol. 6, B. Meltzer and D. Michie (eds.), American Elsevier Publ. Co., New York 1971.
- Pohl, I., "Practical and Theoretical Considerations in Heuristic Search Algorithms," Heuristic Theory Memo HP-75, Computer Science Department, University of California at Santa Cruz, 1975.
- Pohl, I., "Practical and Theoretical Considerations in Heuristic Search Algorithms," in Machine Intelligence 8 (E. Elcock and D. Michie, eds.), Ellis Horwood Ltd., Chichester England 1977.
- Powers, G., et. al., "Optimal Strategies for the Chemical and Enzymatic Synthesis of Bihelical Deoxyribonucleic Acids," J. American Chemical Soc., Vol. 97, 1975, p. 875.
- Raphael, B., The Thinking Computer: Mind Inside Matter, W. H. Freeman & Co., San Francisco 1976.
- Rendell, L., "A Locally Optimal Solution of the Fifteen Puzzle Produced by an Automatic Evaluation Function Generator," Research Report CS-77-36, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, December 1977.
- Rosenfeld, A., R. Hummel and S. Zucker, "Scene labelling by relaxation operations," IEEE Trans. on Systems, Man, and Cybernetics SMC-6, 1976, pp. 420-433.
- Ross, R., Adaptive Aspects of Heuristic Search, Ph.D. thesis, University of Edinburgh, 1973.
- Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers," In E. A. Feigenbaum and Julian Feldman (eds.), Computers and Thought, McGraw-Hill Book Co., New York 1963, pp. 71-105.
- Schofield, P., "Complete Solution of the Eight Puzzle," in Machine Intelligence 1, N. Collins and D. Michie (eds.), American Elsevier Publ. Co., New York 1967.
- Sedgewick, "Quicksort," Dept. of Computer Science, Stanford Univ., report CS-75-492, May 1975.

- Selby, S. (ed.), *Standard Mathematical Tables*, 19th ed., The Chemical Rubber Co., Cleveland 1971.
- SIGART 77. Short notes by D. Cahlander, H. Berliner, and D. Michie, *SIGART Newsletter*, no. 62, April 1977, pp. 8-11.
- Simon, H.A., *Sciences of the Artificial*, MIT Press, Cambridge, MA. 1969.
- Smith, A. R., "Word Hypothesisization for Large-Vocabulary Speech Understanding Systems," Technical Report, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., October 1977.
- Spiegel, M.R., *Statistics*, Schaum's Outline Series, McGraw-Hill Book Co., New York 1971.
- Swinehart, D., and B. Sproull, "SAIL," Stanford AI Project Operating Note No. 57.2, January 1971.
- Sugar, L., "An Experimental Evaluation of Chess Playing Heuristics," Report CSRG-63, Computer Systems Research Group, University of Toronto, December 1975.
- Sussman, G.J., and McDermott, D.V., "Why Conniving is Better than Planning," *Artificial Intelligence Memo*. No. 255A, MIT, 1972.
- Sussman, G. J., and Stallman, R. M., "Forward Reasoning and Dependency-Directed Backtracking in a System for Computer-Aided Circuit Analysis," *Artificial Intelligence*, Vol. 9, 1977, pp. 135-196.
- Tarjan, R.E., "Solving Path Problems on Directed Graphs," Dept. of Computer Science, Stanford University, report STAN-CS-75-528, November 1975.
- Vanderbrug, G., "Problem Representations and Formal Properties of Heuristic Search," *Information Sciences*, Vol. 11, No. 4, 1976.
- Waltz, D.E., "Generating Semantic Descriptions From Drawings of Scenes with Shadows," MAC-AI-TR-271, MIT 1972.
- Waltz, D., "Understanding Line Drawings of Scenes with Shadows," In P. Winston (ed.) *The Psychology of Computer Vision*, McGraw-Hill Book Co., New York 1975, pp. 19-91.
- Weide, B., "A Survey of Analysis Techniques for Combinatorial Algorithms," *Computing Surveys*, Vol. 9, No. 4, December 1977.
- Wells, M.B., "Applications of a Language for Computing in Combinatorics," *Proc. IFIP Congress 65*, Vol. 2, 1965, pp. 497-498.
- Whitehead, E., "Enumerative Combinatorics," Courant Inst. of Math. Sciences Tech. Report, New York University.
- Wickelgren, W., *How to Solve Problems*, W. H. Freeman & Co., San Francisco 1974.

- Winston, P., Artificial Intelligence, Addison-Wesley Publ. Co., Reading Mass. 1977.
- Woods, W., "Shortfall Scoring Strategies for Speech Understanding Control," in Speech Understanding Systems: Quarterly Technical Progress Report No. 6, Bolt Beranek and Newman report No. 3303, 1976.
- Yu, V. L., et al., "Antimicrobial Selection for Meningitis by a Computerized Consultant -- A Blinded Evaluation by Infectious Disease Experts," submitted to Annals of Internal Medicine, August 1978 (1978a).
- Yu, V. L., et al., "Evaluating the Performance of a Computer-Based Consultant," Heuristic Programming Project Memo HPP-78-17, Dept. of Computer Science, Stanford Univ., September 1978 (1978b).
- Zucker, S., "Relaxation Labelling and the Reduction of Local Ambiguities," Report TR-451, Computer Science Dept., University of Maryland, March 1976.
- Zucker, S., E. Krishnamurthy and R. Haar, "Relaxation Processes for Scene Labelling: Convergence, Speed, and Stability," Report TR-477, Computer Science Dept., University of Maryland, August 1976.

Appendix A: Glossary of terms and symbols

For convenience, we list various terms, symbols, abbreviations and acronyms appearing throughout the text. This appendix is organized into three sections, corresponding to Chapters 2, 3, and 4, respectively. The parenthesized page number to the right of each entry identifies the page on which the symbol or term first appears.

Glossary for Chapter 2

- Q a problem graph Q is any finite, directed, strongly connected graph having no loops and no parallel edges. (p. 24)
- (s_r, s_g) denotes a problem instance of a problem graph Q , consisting of an initial node or root node s_r and a goal node s_g . (p. 25)
- $U(Q)$ the set of all problem instances (s_r, s_g) of a problem graph Q . (p. 25)
- $h(s_i, s_j)$ denotes the minimum distance between nodes s_i and s_j of a problem graph Q . (p. 25)
- $\mathbb{R}^+$ the non-negative real numbers (p. 26)
- $F(s) = (1-W) * G(s) + W * K(s)$ the evaluation function used by A^* to score a node s . W is a real on the interval $[0,1]$, $G(s)$ is the distance from node s to the initial node s_r , and $K(s)$ is a heuristic estimate of the distance in the problem graph Q from node s to the goal node s_g . ($K(s)$ is any function from $U(Q)$ to the non-negative reals. (p. 28)
- $K_1(s), K_2(s), K_3(s)$ three particular $K(s)$ functions for the Eight puzzle (pp. 29)
- $X(Q, K, W, s_r, s_g)$ denotes the number of executions of step 4 of A^* before search terminates, for the case of problem instance (s_r, s_g) of problem graph Q , using heuristic function K and weight value W . (p. 29)
- $P(Q, K, W, s_r, s_g)$ denotes the length of the solution path found under the same conditions. (p. 29)
- $L(Q, K, W, s_r, s_g) = P(Q, K, W, s_r, s_g) / h(s_r, s_g)$ (p. 30)
- optimal a particular A^* search is optimal if and only if only nodes along the solution path are expanded and the solution path found is of minimal length, i.e., if and only if $X(s_r, s_g) = P(s_r, s_g) = h(s_r, s_g)$. (p. 30)

$XMEAN(Q, K, W, N)$ denotes the simple arithmetic mean of the values of $X(Q, K, W, s_r, s_g)$ over all problem instances (s_r, s_g) in the set $U(Q)$ such that $h(s_r, s_g) = N$. (p. 33)

$XMAX(Q, K, W, N)$ and $XMIN(Q, K, W, N)$ are defined similarly. (p. 33)

$LMEAN(Q, K, W, N)$ is defined in terms of $L(Q, K, W, s_r, s_g)$ exactly as $XMEAN$ is defined in terms of X . (p. 33)

$LMIN(Q, K, W, N)$ and $LMAX(Q, K, W, N)$ are defined similarly. (p. 33)

$KMIN(Q, K, N)$ is defined to be the minimum value of $K(s_i, s_j)$ over all node pairs (s_i, s_j) in $U(Q)$ such that $h(s_i, s_j) = N$. (p. 25)

Functions $KMAX(Q, K, N)$ and $KMEAN(Q, K, N)$ are defined similarly. (p. 25)

$LEV(Q, K, W, s_r, s_g, i)$ denotes the number of nodes occurring at level i in the search tree as it exists when A^* terminates. (p. 43)

$RUN(Q, K, W, s_r, s_g)$ denotes the mean "run length" of the search, i.e., the number of nodes expanded, divided by one plus the number of "hops" that occur when the next node expanded is not a son of the last node expanded. (p. 43)

$LEVMEAN(Q, K, W, N, i)$ and $RUNMEAN(Q, K, W, N)$ are defined as above. (p. 43)

Glossary for Chapter 3

$T(M, N)$ denotes the uniform tree of branching factor M and unbounded depth, with one distinguished node at level N (called the "goal" node). The root node is at level 0. M is a positive integer. (See Figure 3.1-1.) (p. 77)

s denotes a node in $T(M, N)$ (p. 77)

$g(s)$ denotes the level of node s in $T(M, N)$ (p. 77)

$p(s)$ denotes the level of the deepest common ancestor of node s and the goal node. Call this the "depth of divergence" of node s . (p. 77)

u_i denotes the node at level i on the (unique minimal length) solution path from the root to the goal. Thus u_0 is the root node and u_N is the goal node. We refer to the node $u_{p(s)}$ as the "node of divergence" of a node s . (p. 77)

$r(s) = N - p(s)$, i.e., the distance from the node of divergence of s to the goal (p. 77)

$y(s) = g(s) - p(s)$, i.e., the distance from the node of divergence of s to s (p. 77)

- $v_s(i)$ denotes the node at level i on the path from the root to s (p. 77)
- SP A node s is SP iff it is a node u_i on the solution path (p. 77)
- NSP A node s is NSP iff it is not SP (p. 77)
- $h(s)$ distance from node s to the goal node. $h(s) = g(s) + r(s)$ (p. 78)
- $R_{M, N}(i)$ the set of those nodes in $T(M, N)$ that are at distance i from the goal, i.e., for which $h(s) = i$ (p. 78)
- $\mathbb{R}^+$ the non-negative real numbers (p. 78)
- $K(s)$ any function from the nodes of $T(M, N)$ to the non-negative reals. $K(s)$ is the heuristic estimate of distance from node s to the goal. (p. 82)
- W a real-valued weighting coefficient on the interval $[0,1]$ (p. 82)
- $F(s)$ The evaluation function used by A* to score each node. We assume the form $F(s) = (1-W)*g(s) + W*K(s)$ (p. 78,80,82)
- KS the set of all functions K from the nodes of $T(M, N)$ to the non-negative reals (p. 82)
- $KMIN(M, N, K, i)$ a function bounding the distance estimates of a heuristic function K . For any function $K \in KS$, $KMIN(M, N, K, i)$ equals the smallest value of $K(s)$ for any $s \in R_{M, N}(i)$. (p. 82)
- $KMAX(M, N, K, i)$ like $KMIN$, except bounding $K(s)$ values from above instead of from below. (p. 82)
- KB denotes the set of all functions from the natural numbers to the non-negative reals. Every $KMIN$ and $KMAX$ function belongs to KB. (p. 82)
- $\mathbb{N}$ the non-negative integers (p. 82)
- KB^* the set of all pairs $\langle KMIN, KMAX \rangle$ such that $\forall i \ KMIN(i) \leq KMAX(i)$. Note that $KB^* \subset KB \times KB$. (p. 82)
- $KWORST(KMIN, KMAX, s) = \begin{cases} KMAX(h(s)) & \text{if } s \text{ is SP} \\ KMIN(h(s)) & \text{if } s \text{ is NSP} \end{cases}$ (p. 84)
- $XWORST(M, KMIN, KMAX, W, N)$ denotes the number of nodes of $T(M, N)$ expanded during A* search using evaluation function $F(s) = (1-W)*g(s) + W*KWORST(KMIN, KMAX, s)$ (p. 84)
- $YMAX(KMIN, KMAX, W, r)$ denotes the largest distance away from from the goal path of

- expanded nodes using $\langle KMIN, KMAX \rangle$ under worst case conditions (see Definition 3.1-4a) (p. 85)
- IM a KMIN or KMAX function for which $F(s)$ values increase monotonically with distance from the goal (see Lemmas 3.2-1 and 3.2-2) (pp. 88-89)
- DM a KMAX function for which $F(s)$ values decrease monotonically with distance from the goal (see Lemma 3.2-3) (pp. 88-89)
- linearly bounded any $\langle KMIN, KMAX \rangle$ such that $KMIN(i) = a \cdot i$ and $KMAX(i) = b \cdot i$, where a and b are reals such that $0 \leq a \leq b$ (p. 92)
- $\delta(KMIN, KMAX, i)$ the relative error function corresponding to a given $\langle KMIN, KMAX \rangle$ function pair (see Definition 3.3-1). (p. 96)
- $\alpha(KMIN, KMAX, i)$ the mean value function corresponding to a given $\langle KMIN, KMAX \rangle$ function pair (see Definition 3.3-1). (p. 96)

Glossary for Chapter 4

- SAP Acronym for Satisficing Assignment Problem (p. 146)
- S denotes a SAP (p. 146)
- N denotes the number of problem variables of a SAP (p. 146)
- x_i denotes a problem variable of a SAP (p. 146)
- p.v. abbreviation for problem variable
- k_i denotes the number of candidate values of problem variable x_i (p. 146)
- R_i denotes the set of candidate values of problem variable x_i (p. 146)
- v_{ij} denotes a candidate value of problem variable x_i (p. 146)
- c.v. abbreviation for candidate value
- P_{ij} denotes the constraint relation between problem variables x_i and x_j . Note that $P_{ij} \subseteq R_i \times R_j$ (p. 146)
- proper a constraint relation P_{ij} is proper if and only if $P_{ij} \subset R_{ij}$ (p. 146)
- A denotes an assignment of candidate values to problem variables; $A \in U = R_1 \times R_2 \times \dots \times R_N$ (p. 146)

- $A(i)$ the i 'th element of assignment A , i.e., the candidate value assigned to problem variable x_i under assignment A (p. 146)
- solution** An assignment A is a solution if and only if for every i and j such that $1 \leq i < j \leq N$, $(A(i), A(j)) \in P_{ij}$ (p. 147)
- complete consistency graph** A SAP S has a complete consistency graph if and only if $P_{ij} \subseteq R[ij]$ for all i and j . (p. 148)
- incomplete consistency graph** A SAP S has an incomplete consistency graph if and only if it does not have a complete consistency graph. (p. 148)
- pair-test** a unit of computation; for the 8-queens problem, one pair-test determines whether a queen on a specified square attacks a queen on another specified square. (p. 150)
- T_f, T_a number of pair-tests executed before finding a first solution (T_f), or before finding all solutions (T_a) (p. 153)
- D_f, D_a number of distinct pair-tests executed (as for T_f and T_a) (p. 153)
- M_f, M_a redundancy ratio; $M = T / D$ (p. 153)
- $T_{\min}$ minimum number of pair-tests executed to find a first solution for a SAP having at least one solution. $T_{\min}(N) = N(N-1)/2$ for SAPs having N problem variables. (p. 153)
- $D_{\max}$ total number of distinct pair-tests for a SAP. (p. 154)
- SAS** denotes the size of the assignment space; $SAS = \prod_{1 \leq i \leq N} k_i$ (p. 154)
- BACKMARK** A new algorithm valid for all SAPs. Defined in SAIL code. (p. 165)
- BACKJUMP** A new algorithm valid for all SAPs. Defined in SAIL code. (p. 170)
- DEELEV** A new algorithm valid for all SAPs, a generalization of the DEEB algorithm that backtracks to level i before commencing DEEB search. (p. 154)
- N-similar, N- k_i -similar, N- k_i -L-similar** Certain relations that may hold between two SAPs (pp. 177)
- L** a measure of degree of constraint of a SAP (p. 177)
- π_i A permutation of the candidate values of problem variable x_i , i.e., a permutation of the set R_i (p. 182)
- valid** An algorithm A is valid for all SAPs as in Definition 5-1 if it always finds every solution that exists for every SAP, and never reports as a solution an assignment that is not a solution. (p. 184)

Appendix B: Extensions of Present Experiments and Analysis

As discussed in Section 1.5, the exploratory nature of this dissertation precluded as deep an investigation of some matters as might be desired, in deference to reserving effort for other topics as well. Consequently, there are numerous points in the dissertation at which the possibility of additional results is apparent.

This appendix suggests certain possible extensions, both experimental and analytic, to the results reported in this dissertation. This appendix is divided into three sections, concerning Chapters 2, 3, and 4, respectively. Most of the extensions concern well-defined tasks, such as performing a certain experiment or attempting to prove that a particular conjecture is true. We include also a few less definite tasks, such as attempting to devise an algorithm having certain properties. Each task is identified by a code such as E2-1, which denotes the first enumerated extension concerning Chapter 2.

Extensions concerning Chapter 2

E2-1: Analogous experiments for different problems

All of the data reported in Chapter 2 concern a single problem graph, that of the 8-puzzle. For the purpose of determining limits to the generality of the results, it would be useful to obtain comparable results for other problem graphs. For example, the 8-puzzle, 15-puzzle, 24-puzzle, ..., (p^2-1) -puzzle problem graphs differ, to a first approximation, only in size. In particular, heuristic functions K_1 , K_2 , and K_3 can be applied to each of these graphs. (K_3 is generalized in an obvious way for $p > 3$.) Hence case study results spanning members of this problem family would show how the performance of best-first search depends on size of the problem, other things (including the heuristic function) being equal.

Whereas the 15-puzzle and 24-puzzle graphs merit consideration for case study experiments by virtue of being intuitively very similar to the 8-puzzle graph, the peg solitaire problem [Jackson 1974, p. 114] merits consideration because it differs from the 8-puzzle graph in being a directed graph, and in having nodes of degree one (i.e., "dead ends"). Like the 8-puzzle, the peg solitaire problem also has generalized forms that vary in the size and geometric shape (e.g., square, triangle, hexagon) of the board on which the game is played.

E2-2: Varying W dynamically during a single search

Figures in Section 2.3 show that increasing W decreases $XMEAN$ for values of N large with respect to the diameter of the graph, but show also that $XMEAN$ actually increases for small and mid-sized N as W increases. This suggests a variation of A^* by which the value of W used during search of a given problem instance be varied dynamically during that search, taking an initial large value and decreasing (perhaps monotonically with number of expansions) as a function of the estimated distance to the goal (e.g., using the value $K(s)$ as estimate of the latter). Design and perform an experiment to discover what decrease in $XMEAN(N)$, if any, is realized by such a scheme.

E2-3: Comparison of ordered depth-first search with best-first search

The $RUNMEAN$ data shown in Figure 2.5-3 suggest the following comparison between the performance of A^* search and that of ordered depth first search (ODFS). In ODFS a node is selected for expansion at iteration t from among the sons of the node expanded at iteration $t-1$, whereas in A^* the set of candidates for expansion at iteration t consists of all distinct nodes OPENed during iterations $1, 2, \dots, t-1$ and not yet expanded. Hence A^* can "hop around" the extant search tree whereas ODFS does not construct such a tree. Any ordering function of the form $F(s) = (1-W) G(s) + W K(s)$ used by A^* can be used by ODFS to select the son to expand next, hence A^* and ODFS are alternative search algorithms. By definition, the length of the solution path found, if any, by ODFS equals the number of nodes expanded, i.e., $P(Q, K, W, s_r, s_g) = X(Q, K, W, s_r, s_g)$. (The "if any" qualification is necessary for the case that a circuit is found, i.e., the selected node has been expanded already.) Hence it may be the case that paths found by ODFS will be longer than those found by A^* under the same conditions. Intuition suggests the possibility that $X_{ODFS}(Q, K, W, s_r, s_g) < X_{A^*}(Q, K, W, s_r, s_g)$, at least for some values of the parameters (since ODFS does not hop around a search tree). Illustrating with the case of heuristic function K_1 , the rationale is that K_1 rarely follows a path segment for more than one step before hopping to a different portion of the tree to pursue a partial path previously abandoned (as indicated by the $RUNMEAN$ evidence); wasted expansions occur prominently in middle depths of the search tree, not near the goal (as indicated by the $LEVMEAN$ evidence): ordered depth first search follows a single path until the goal is eventually reached, avoiding the aforementioned waste (although possibly succumbing to other kinds). This motivates replicating all of the experiments reported in Chapter 2 using ODFS as the search schema instead of A^* , and comparing the corresponding values of $XMEAN(K, W, N)$ and $LMEAN(K, W, N)$ with those reported in Chapter 2. See [Winston 1977, pp. 93-98] and [Wickelgren 1974, pp. 67-90] for informal discussions of the limitations of hill climbing as applied to state space problems.

E2-4: $LEVMEAN(N)$ and $RUNMEAN(N)$ for arbitrary W

$LEVMEAN$ data were reported in Section 2.5 only for the illustrative case $N = 20$, $W = .5$. Similarly, $RUNMEAN$ data were reported only for the case $W = .5$. Data corresponding to those shown in Figures 2.5-2, 2.5-4, and 2.5-5 were in fact collected for all values of $2 \leq N \leq 26$ and for $W = .1, .2, \dots, 1.0$. Using these data, it would be interesting to relate changes in $XMEAN(N)$ as W increases with corresponding changes

in LEVMEAN(N) and in RUNMEAN(N). Such an investigation of the "internal" behavior of A* might help to explain the observed XMEAN(N) performance, particularly that reported in Section 2.3.

E2-5: Individual vs. aggregate performance comparisons

Two problem instances are cited in Section 2.2 that provided conflicting evidence about Nilsson's hypothesis regarding the relative merits of K_2 and K_3 . Figures 2.2-1, 2.2-3, 2.2-4, and 2.2-5 provide evidence aggregated over all problem instances of distance N in the sample set. It may also be interesting to compare the performance of K_3 against that of K_2 for each problem instance individually, i.e., compare $X(K_3, .5, s_r, s_g)$ with $X(K_2, .5, s_r, s_g)$ and similarly $P(K_3, .5, s_r, s_g)$ with $P(K_2, .5, s_r, s_g)$ for each (s_r, s_g) . Specifically, for each value of N what percentage of problem instances of distance N in the sample set fall in each of the following disjoint categories?

- a) $X(K_3) > X(K_2)$
- b) $X(K_3) < X(K_2)$ and $P(K_3) > P(K_2)$
- c) $X(K_3) < X(K_2)$ and $P(K_3) = P(K_2)$

Extensions Concerning Chapter 3

E3-1: Worst case "radius of optimality"

Given a heuristic function $\langle KMIN, KMAX \rangle$, let the worst case radius of optimality of $\langle KMIN, KMAX \rangle$ at W be defined as the largest positive integer k such that for all integers N such that $1 \leq N \leq k$,

$$XWORST(M, KMIN, KMAX, W, N) = N$$

Derive a formula, as a function of W , for the worst case radius of optimality of all $\langle KMIN, KMAX \rangle$ that are: (a) linearly bounded; (b) IM-tight-underestimating; (c) IM. In particular, for each of the cases (a), (b), and (c), does the radius of optimality vary monotonically with W ?

E3-2: Number of nodes expanded at level i in search tree

Let $LEVWORST(M, KMIN, KMAX, W, N, i)$ denote the number of nodes occurring at level i in the tree $T(M, N)$ that are expanded under the conditions of Theorem 3.1-2. By this definition,

$$\sum_{0 \leq i < \infty} LEVWORST(M, KMIN, KMAX, W, N, i) = XWORST(M, KMIN, KMAX, W, N)$$

Derive formulas for LEVWORST for the cases that $\langle KMIN, KMAX \rangle$ is: (a) linearly bounded; (b) IM-tight-underestimating; (c) IM.

Note that the LEVMEAN data for three 8-puzzle heuristics reported in Figure 2.5-2 suggest that for a given search the value of i for which LEVMEAN is largest is about $N/2$. For given M , $KMIN$, $KMAX$, W , and N , what is the value of i that maximizes $LEVWORST(M, KMIN, KMAX, W, N, i)$?

E3-3: Cost grows monotonically with $KMAX(i) - KMIN(i)$

In practical situations it is sometimes necessary to choose between two candidate heuristic functions the one that minimizes cost. Monotonicity theorems such as those in Chapter 3 are relevant to such questions. For example, Theorem 3.2-5 showed for any IM heuristic functions K_1 and K_2 having identical $KMAX$ functions and $KMIN_1(i) \leq KMIN_2(i)$ for all i , that $XWORST(M, KMIN_1, KMAX, W, N) \geq XWORST(M, KMIN_2, KMAX, W, N)$ for all M , N , and $0 \leq W < 1$. Similarly, Theorem 3.2-6 showed that if K_1 and K_2 have identical $KMIN$ functions and $KMAX_1(i) \leq KMAX_2(i)$ for all i , then worst case cost using K_1 never exceeds that using K_2 . Corollary 3.2-7 combined these two theorems.

Now assume simply that $KMAX_1(i) - KMIN_1(i) \leq KMAX_2(i) - KMIN_2(i)$ for all i , and K_1 and K_2 are IM. For which such K_1 and K_2 and W is it the case that $XWORST(M, KMIN_1, KMAX_1, W, N) \geq XWORST(M, KMIN_2, KMAX_2, W, N)$ for all M and N ?

E3-4: Optimal weighting

Theorem 3.4-1 determined for all IM-tight-underestimating heuristic functions that the weighting value $W = .5$ minimizes $XWORST$ for all M and N . Subsequently in Section 3.4 we showed that this does not generalize to the entire class of IM heuristic functions: we determined the locus of linearly bounded heuristic functions for which $W = 1.0$ gives as good or better performance as $W = .5$. Determine criteria identifying the subsets of IM heuristic functions for which: (a) $W = .5$ minimizes cost; (b) $W = 1.0$ minimizes cost.

Extensions Concerning Chapter 4

E4-1: Analogous experiments with different problems

Experiments analogous to those reported in Chapter 4 concerning the N-Queens SAPs could be performed for other "particular situation" SAPs. Each of the following SAPs contrasts with the N-Queens SAPs in a different way:

- a) Repeat the experiments of Sections 4.1.2, 4.3, and 4.4.2 for "N-Queen-

knights" SAPs, and similarly for "N-rook-knights" SAPs, for "N-rook-king-knights" SAPs, for "N-bishop-knights" SAPs, and so on. The "N-Queen-Knights" SAPs are defined like the N-Queens SAPs, except that the "chess pieces" in the former move either as queens or as knights. The other variations are defined similarly. (Like the N-Queens SAPs, all of these have complete consistency graphs.) These SAPs differ from N-Queens SAPs and from each other only in degree of constraint (L), to a first approximation. Hence experimental results from all of these could be combined into plots of the form given in Section 4.4.3 (i.e., for each value of N , each problem named above contributes data for a different value of L). Another objective is to compare the relative effects on cost of varying L vs. varying size (N): hold each constant and vary the other.

- b) Measure the performances of the various algorithms tested in Chapter 4 for generalized Instant Insanity, which is defined like the familiar version, but using N cubes colored randomly with N distinct colors. These SAPs vary in size, but only in N : the k_i are constant with N ($k_i = 24$ rotations of a cube). This contrasts with N-Queens SAPs, for which both N and the k_i values vary together. (Like the N-Queens SAPs, the Instant Insanity SAPs have complete consistency graphs.)
- c) Measure the performance of DEEB for the map coloring problem defined in Section 4.5.1, and compare the results with those for the other algorithms reported in that section. Extend the experiments to other maps. (Map coloring SAPs have incomplete consistency graphs.)
- d) Replicate Waltz' experiments with line drawing assignment problems, recording performances for each of the four algorithms tested here. A simpler version of Waltz' line drawing problem is defined using as candidate values the legal labelings given on pp.35-38 of [Waltz 1972]. (These line drawing SAPs have incomplete consistency graphs.)
- e) For each of the problems listed above, generate a sample of randomized versions of the problem in the manner of Section 4.4.2. Compare the randomized versions to the familiar versions by executing experiments analogous to those reported in Section 4.4.2.

E4-2: Study problem structure using parameterized random problems

- a) Partition each equivalence class defined by the $N-k_i-L$ relation (defined in Section 4.4.1) into subclasses such that SAPs S and S' are $N-k_i-L_{ij}$ -equivalent if and only if $L_{ij} = L'_{ij}$ for all i and j , where $L_{ij} = |P_{ij}| / (k_i * k_j)$. Then repeat the experiment in Section 4.4.3, using randomly generated samples of SAPs that are $N-k_i-L_{ij}$ -similar to the N-Queens SAPs. As in that section, the objective is to determine whether this "finer grain" partitioning is such that an N-Queens SAP is typical of the elements in the equivalence class to which it belongs.
- b) Partition the $N-k_i-L$ equivalence class to which the 8-queens SAP belongs

into subclasses such that the 8-Queens SAP is the only element in one subclass; another subclass contains all those SAPs identical to 8-Queens (under some c.v. ordering) except that one element of one P_{ij} relation is deleted and another such element is added to a possibly different relation P_{mn} ; another subclasses contains all those SAPs differing from 8-Queens by two (in general c) such "link changes". This is another metric for "structure", in which a SAP is an N -partite graph and we randomly change c edges. How does T_f vary with c ?

E4-3. Dependence of cost on degree of constraint

- a) Repeat the experiments of Section 4.4.3 for the cases $L = .55, .59, .61$, and $.65$. Plot Figures 4.4.3-1, 4.4.3-2, and 4.4.3-3 including the new data so obtained. These finer-grained results may indicate how sharp the apparent peak at approximately $L = .6$ is.
- b) Repeat the experiments of Section 4.4.3 for cases other than $N = 10 = k_1 = \dots = k_{10}$.

E4-4. Test BACKMARK for 75-queens and 100-queens

Section 4.3 reports applications of algorithm BACKMARK to N -queens SAPs for N as large as 50. Extend these experiments to $N = 75$ and $N = 100$. Using data from these experiments, extend Figure 4.2-5 for the cases $N = 75$ and $N = 100$.

E4-5. DEELEV(i) using BACKMARK instead of BACKTRACK

Section 4.2.3 defines an algorithm DEELEV(i) that combines DEEP and BACKTRACK. Define a new version of DEELEV(i) substituting BACKMARK for BACKTRACK (the substitution is routine). Repeat the experiments of Section 4.2.3 using this new algorithm and compare the results with those plotted in Section 4.2.3.

E4-6. Solution distribution for random candidate value ordering

Repeat the experiments of Section 4.5.2 for instances of the 5-Queens SAP having random candidate value ordering instead of the "obvious" candidate value ordering assumed in that section. Compare the former with the latter by adding plots of the results obtained for the former to those obtained for the latter in Figure 4.5.2-1. Repeat this procedure for the 7-Queens SAP and the 8-Queens SAP.

E4-7. Relation between efficiency and number of solutions

In section 4.1.2, we observed of the N -Queens problems that the cost of finding a solution (i.e., T_f) seems to be related to the total number of solutions possessed by the problem instance (Figure 4.1.2-4). The limited $\text{prod}(N)$ data reported in that section has only suggestive value. However, these data could be supplemented by analogous data for randomly generated problem instances, if the total number of solutions possessed by a randomly generated problem instance were known. Select a subset of the random- N -queens SAPs tested in Section 4.4.2, including several problem

instances for each value of N . For each such problem instance, find all solutions, thereby determining the number of solutions possessed by that instance. For each such problem instance, multiply the number of solutions by the T_f value measured during the experiment of Section 4.4.2. Plot the resulting products against value of N as a scatter plot superimposed on Figure 4.1.2-4.

Appendix C

Tabulation of Experimental Data Plotted in the Figures

The numerical values plotted in the figures of Chapter 2, Chapter 4, and Section 3.5 constitute the actual results of the experiments reported therein. This appendix tabulates each value plotted in most of these figures. This appendix is organized into three sections, pertaining to Chapters 2, 3, and 4, respectively. Note that trailing zeros to the right of the decimal point are not necessarily significant (due to variations in hand data entry). To save space, tables not completed on one page continue on the next.

Tabulated Data for Figures in Chapter 2

Figure 2.2-1 Mean, maximum, and minimum no. of nodes expanded vs. distance to goal
Re search of 8-puzzle using heuristic function K_1 , using $W = .5$
48 samples per data point, 768 samples total

N	XMEAN	XMIN	XMAX Breadth-first	
1	1.000	1	1	1.000
2	2.000	2	2	2.600
3	3.000	3	3	6.800
4	4.125	4	5	14.050
5	5.425	5	7	26.450
6	7.275	6	10	44.925
7	9.350	7	14	81.025
8	12.625	8	20	128.580
9	20.025	9	34	220.200
10	29.125	14	50	380.630
11	47.250	22	72	648.230
12	71.475	42	99	950.900
13	114.480	73	174	1578.300
14	168.680	114	248	2673.200
15	271.600	185	426	3805.400
16	426.400	213	662	
17	746.750	413	1030	
18	1061.700	666	1361	
19	1707.900	1486	2305	
20	2659.400	2061	3385	

Figure 2.2-3 Analogous to Figure 2.2-1, but using heuristic function K_2
895 problem instances

N	XMIN	XMEAN	XMAX Breadth-first	
1	1		1	1.000

2	2	2.000	2	2.600
3	3	3.000	3	6.800
4	4	4.000	4	14.050
5	5	5.100	6	26.450
6	6	6.150	8	44.925
7	7	7.500	12	81.025
8	8	9.100	17	128.580
9	9	11.575	29	220.200
10	10	14.375	32	386.630
11	11	18.475	39	648.230
12	12	21.500	59	950.900
13	13	35.325	90	1578.300
14	14	37.125	93	2673.200
15	15	55.825	146	3805.400
16	16	79.400	299	
17	17	119.230	307	
18	22	161.950	380	
19	22	237.450	497	
20	20	267.050	600	
21	61	351.600	951	
22	47	422.670	1051	
23	100	691.470	1622	
24	107	1042.900	2241	
25	273	1552.400	2794	
26	379	1958.500	4251	

Figure 2.2-4 Analogous to Figure 2.2-1, but using heuristic function K_3
895 problem instances

N	XMIN	XMEAN	XMAX Breadth-first
1	1		1 1.000
2	2	2.000	2 2.600
3	3	3.000	3 6.800
4	4	4.000	4 14.050
5	5	5.050	6 26.450
6	6	6.125	8 44.925
7	7	8.825	22 81.025
8	8	8.275	10 128.580
9	9	10.375	26 220.200
10	10	13.650	69 386.630
11	11	14.775	40 648.230
12	12	25.400	111 950.900
13	13	22.350	67 1578.300
14	14	26.550	86 2673.200
15	15	36.675	139 3805.400
16	18	32.250	158
17	17	55.425	163
18	18	53.900	210
19	19	66.300	244
20	20	62.100	201
21	21	59.900	161
22	22	92.533	234
23	26	93.867	402
24	32	81.240	186
25	27	99.917	382
26	33	57.375	89

Figure 2.2-5 Mean number of nodes expanded vs. depth of goal,
Data from Figures 2.2-1, 2.2-3, and 2.2-4
768 + 895 + 895 = 2358 algorithm executions

N	K ₂	K ₁	K ₃ Breadth-first	
1	1.000	1.000	1.000	1.000
2	2.000	2.000	2.000	2.600
3	3.000	3.000	3.000	6.800
4	4.000	4.125	4.000	14.050
5	5.100	5.425	5.050	26.450
6	6.150	7.275	6.125	44.925
7	7.500	9.350	8.825	81.025
8	9.100	12.625	8.275	128.580
9	11.575	28.025	10.375	228.200
10	14.375	29.125	13.600	386.630
11	18.475	47.250	14.775	648.230
12	21.500	71.475	25.400	958.900
13	35.325	114.480	22.350	1578.300
14	37.125	168.680	28.550	2673.200
15	55.825	271.600	36.675	3805.480
16	79.400	426.400	32.250	
17	119.230	746.750	55.425	
18	161.950	1061.700	53.900	
19	237.450	1767.900	66.300	
20	267.050	2659.400	62.100	
21	351.600		59.900	
22	422.670		92.533	
23	691.470		93.867	
24	1042.900		81.248	
25	1552.400		99.917	
26	1958.500		57.375	

Figure 2.2-6 Length of solution path vs. depth of goal
heuristic function K₃, using W = .5
(L = 1 if minimal length solution path is found)
895 algorithm executions

N	L _{MIN}	L _{MEAN}	L _{MAX}
2	1.000	1.000	1.000
3	1.000	1.000	1.000
4	1.000	1.000	1.000
5	1.000	1.000	1.000
6	1.000	1.000	1.000
7	1.000	1.000	1.000
8	1.000	1.000	1.000
9	1.000	1.000	1.000
10	1.000	1.025	2.000
11	1.000	1.014	1.364
12	1.000	1.125	2.333
13	1.000	1.069	1.769
14	1.000	1.079	2.143
15	1.000	1.160	1.933
16	1.000	1.100	1.625
17	1.000	1.194	1.824

18	1.888	1.175	1.889
19	1.888	1.232	1.842
20	1.888	1.155	1.988
21	1.888	1.127	1.762
22	1.888	1.289	1.727
23	1.888	1.215	1.689
24	1.888	1.193	1.583
25	1.888	1.193	1.328
26	1.888	1.154	1.231

Figure 2.3-1 Mean number of nodes expanded vs. depth of goal
Heuristic function K_1 with different weight values
688 to 895 algorithm executions per value of W

N	W = 1.8	W = .5	W = .2	W = .7 Breadth-first
1	1.888	1.888	1.888	1.888
2	2.888	2.888	2.888	2.888
3	3.888	3.888	4.650	3.888
4	4.588	4.125	7.725	4.125
5	14.388	5.425	12.725	5.425
6	65.375	7.275	22.675	7.488
7	182.358	9.358	38.688	18.225
8	199.188	12.625	65.125	15.358
9	228.188	20.825	118.238	26.175
10	174.238	29.125	191.388	43.458
11	367.638	47.258	311.788	66.588
12	341.858	71.475	506.638	74.858
13	298.938	114.488	792.138	122.458
14	441.458	168.688	1387.888	168.158
15	415.988	271.688	2812.888	285.588
16	384.938	426.488	3363.188	355.258
17	371.258	746.758		718.658
18	433.458	1861.788		728.538
19	488.988	1767.988		985.888
20	398.188	2659.488		1678.188
21	494.288			1791.288
22	476.288			2988.888
23	481.388			
24	526.888			
25	554.688			
26	584.258			

Figure 2.3-2 Analogous to Figure 2.3-1, but for heuristic K_2
648 to 895 algorithm executions per value of W

N	W = 8.5	W = 0.7	W = 1.8	W = 0.2 Breadth-first
1	1.888	1.888	1.888	1.888
2	2.888	2.888	2.888	3.758
3	3.888	3.888	3.888	5.488
4	4.888	4.888	4.888	18.988
5	5.188	5.188	5.188	15.275
6	6.158	6.158	6.258	25.788
7	7.588	7.525	12.558	48.675

8	9.100	9.150	34.525	64.825	128.580
9	11.575	12.125	37.900	103.450	220.200
10	14.375	17.375	57.175	167.300	386.630
11	18.475	29.025	101.780	257.330	648.230
12	21.500	32.000	90.625	385.130	950.900
13	35.325	48.500	111.830	613.600	1578.300
14	37.125	63.725	168.550	909.950	2673.200
15	55.825	119.200	159.350	1417.600	3805.400
16	79.400	133.350	139.730	2257.100	
17	119.230	186.700	203.230	3457.700	
18	161.950	207.030	165.350		
19	237.450	239.700	170.580		
20	267.050	326.630	183.430		
21	351.600	308.270	177.730		
22	422.670	302.170	214.470		
23	691.470	328.270	227.100		
24	1042.900	516.240	211.560		
25	1552.400	404.670	242.580		
26	1958.500	455.630	258.880		

Figure 2.3-3 Analogous to Figure 2.3-1, for heuristic K_3
 These data suggest that increasing W beyond .5 has no apparent effect for K_3
 875 to 895 algorithm executions per value of W

N	$W = .5$	$W = 1.0$	$W = .2$	$W = .7$
1	1.000	1.000	1.000	1.000
2	2.000	2.000	2.000	2.000
3	3.000	3.000	3.575	3.000
4	4.000	4.000	4.600	4.000
5	5.050	5.050	6.400	5.050
6	6.125	6.175	7.750	6.125
7	8.825	13.525	9.900	9.775
8	8.275	8.350	11.625	8.275
9	10.375	13.550	14.875	11.825
10	13.650	22.000	19.250	14.950
11	14.775	19.950	25.975	17.900
12	25.400	38.800	34.375	32.300
13	22.350	28.500	48.975	24.125
14	26.550	32.125	64.425	29.800
15	36.675	39.075	92.300	41.450
16	32.250	34.725	133.630	38.725
17	55.425	54.575	192.750	52.625
18	53.900	60.375	257.200	49.575
19	66.300	59.050	345.150	51.700
20	62.100	67.325	501.030	60.175
21	59.900	76.667	674.070	64.900
22	92.533	84.733	990.870	74.733
23	93.867	87.467	1620.400	74.500
24	51.240	85.920	2350.100	81.800
25	99.917	96.667		82.667
26	57.375	84.375		101.630

Figure 2.3-4 XMEAN vs. W for K_2 , taken from same data as for Figure 2.3-2

W	N = 5	N = 10	N = 15	N = 20	N = 25
.000	26.450	386.600	3805.000		
.100	18.500	218.400	2254.000		
.200	15.275	167.380	1417.000		
.300	7.975	57.750	449.100	3353.000	
.400	5.175	27.425	167.550	1143.200	
.500	5.100	14.375	55.800	267.800	1552.000
.600	5.100	14.600	67.000	259.400	697.300
.700	5.100	17.375	119.200	326.600	404.670
.800	5.100	24.600	168.230	261.200	218.800
.900	5.100	42.700	179.830	252.500	301.300
1.000	5.100	57.175	159.850	183.430	242.600

Figure 2.3-5 Mean no. of nodes expanded vs. depth of goal, 3 heuristics using $W = 1.0$
 Combines data for $W = 1.0$ from Figures 2.3-1, 2.3-2, and 2.3-3
 At $W=1.0$, all 3 K functions have "subexponential" cost. Compare with Figure 2.2-5
 3 * 895 = 2385 algorithm executions

N	K_1	K_2	K_3 Breadth-first
1	1.000	1.000	1.000
2	2.000	2.000	2.000
3	3.000	3.000	6.800
4	4.500	4.000	14.050
5	14.300	5.100	26.450
6	65.375	6.250	44.925
7	102.350	12.550	81.025
8	199.180	34.525	128.580
9	220.100	37.900	220.200
10	174.230	57.175	386.630
11	367.630	101.780	548.230
12	341.050	90.625	950.900
13	298.930	111.830	1578.300
14	441.450	168.550	2673.200
15	415.900	159.850	3805.400
16	304.930	139.430	34.725
17	371.250	203.230	54.575
18	433.450	165.350	60.375
19	400.900	170.580	59.050
20	398.180	183.430	67.325
21	494.200	177.730	76.667
22	476.200	214.470	84.733
23	481.300	227.100	82.467
24	526.800	211.560	85.920
25	554.000	242.580	96.667
26	584.250	258.880	84.375

Figure 2.3-6 Mean length of solution path vs. depth of goal, for various W
 heuristic function K_1
 820 to 895 algorithm executions per value of W

N	$W = 1.0$	$W = .9$	$W = .8$	$W = .7$
1	1.000	1.000	1.000	1.000

2	1.883	1.000	1.000	1.000
3	1.000	1.000	1.000	1.000
4	1.150	1.000	1.000	1.000
5	1.110	1.000	1.000	1.000
6	3.817	1.000	1.000	1.000
7	4.029	1.000	1.000	1.000
8	5.831	1.106	1.019	1.090
9	6.306	1.294	1.061	1.011
10	5.285	1.495	1.075	1.005
11	8.145	1.580	1.122	1.018
12	7.279	1.530	1.071	1.021
13	6.392	1.315	1.088	1.008
14	7.050	1.500	1.100	1.025
15	6.993	1.513	1.136	1.023
16	5.119	1.387	1.109	1.009
17	5.756	1.429	1.147	1.029
18	6.019	1.356	1.081	1.022
19	5.345	1.358	1.095	1.013
20	5.398	1.372	1.085	1.033
21	5.787	1.336	1.105	1.044
22	5.494	1.333	1.088	1.023
23	5.539	1.325	1.087	
24	5.427	1.397		
25	5.707	1.287		
26	5.067	1.327		

Figure 2.3-7 Mean length of solution path vs. depth of goal, for various W heuristic function K_2
 $4 \times 895 = 3180$ algorithm executions

N	$W = 1.0$	$W = .7$	$W = .8$	$W = .9$
2	1.000	1.000	1.000	1.000
3	1.000	1.000	1.000	1.000
4	1.000	1.000	1.000	1.000
5	1.000	1.000	1.000	1.000
6	1.000	1.000	1.000	1.000
7	1.314	1.000	1.000	1.000
8	1.725	1.000	1.000	1.000
9	1.878	1.000	1.006	1.007
10	2.000	1.010	1.025	1.170
11	2.986	1.005	1.150	1.450
12	2.521	1.025	1.140	1.340
13	2.231	1.019	1.170	1.280
14	2.964	1.036	1.275	1.546
15	2.970	1.113	1.360	1.640
16	2.322	1.113	1.270	1.520
17	3.277	1.079	1.320	1.690
18	2.539	1.072	1.260	1.550
19	2.605	1.108	1.297	1.620
20	2.663	1.162	1.347	1.622
21	2.683	1.149	1.280	1.619
22	2.997	1.139	1.345	1.720
23	2.759	1.136	1.296	1.620
24	2.663	1.150	1.350	1.757
25	3.020	1.090	1.270	1.840
26	3.260	1.144	1.336	1.580

Figure 2.3-8 Mean length of solution path vs. depth of goal, for various M heuristic function K_3
 4 * 895 = 3180 algorithm executions

N	M = .5	M = .7	M = .9	M = 1.0
2	1.000	1.000	1.000	1.000
3	1.000	1.000	1.000	1.000
4	1.000	1.000	1.000	1.000
5	1.000	1.000	1.000	1.000
6	1.000	1.000	1.000	1.000
7	1.000	1.000	1.000	1.328
8	1.000	1.000	1.000	1.000
9	1.000	1.000	1.000	1.160
10	1.025	1.060	1.160	1.485
11	1.014	1.155	1.173	1.353
12	1.125	1.300	1.379	1.920
13	1.069	1.119	1.296	1.380
14	1.079	1.221	1.370	1.440
15	1.160	1.267	1.570	1.630
16	1.180	1.281	1.340	1.490
17	1.194	1.391	1.510	1.890
18	1.175	1.333	1.510	1.930
19	1.232	1.324	1.460	1.820
20	1.155	1.313	1.480	1.940
21	1.127	1.416	1.620	2.060
22	1.209	1.439	1.580	2.150
23	1.215	1.357	1.570	2.130
24	1.193	1.370	1.470	2.110
25	1.193	1.407	1.630	2.270
26	1.154	1.231	1.470	2.060

Figure 2.3-9 Mean length of solution path vs. depth of goal, $M = 1.0$
 Comparison of heuristics K_1 , K_2 , and K_3
 (LMEAN = 1 means minimal length solution path was found)
 3 * 895 = 2385 algorithm executions

N	K_1	K_2	K_3
1	1.000	1.000	
2	1.000	1.000	1.000
3	1.000	1.000	1.000
4	1.150	1.000	1.000
5	1.110	1.000	1.000
6	3.017	1.000	1.000
7	4.029	1.314	1.329
8	5.831	1.725	1.000
9	6.306	1.070	1.161
10	5.205	2.000	1.485
11	8.145	2.986	1.359
12	7.279	2.521	1.917
13	6.392	2.231	1.385
14	7.050	2.964	1.443
15	6.993	2.970	1.633
16	5.119	2.322	1.480
17	5.756	3.277	1.891

18	6.019	2.539	1.931
19	5.345	2.605	1.821
20	5.398	2.663	1.943
21	5.787	2.683	2.060
22	5.494	2.997	2.155
23	5.539	2.759	2.130
24	5.427	2.663	2.107
25	5.707	3.020	2.267
26	5.067	3.260	2.058

Figure 2.3-10 Aggregate LMEAN(W). See text
Each data point averages LMEAN over a range of W
More than 26,000 algorithm executions

W	K ₁	K ₂	K ₃
.100	1.000	1.000	
.200	1.000	1.000	1.003
.300	1.000	1.000	1.022
.400	1.000	1.000	1.092
.500	1.000	1.000	1.093
.600	1.001	1.011	1.137
.700	1.013	1.060	1.192
.800	1.064	1.174	1.260
.900	1.278	1.371	1.300
1.000	4.852	2.207	1.560

Figure 2.3-14 Maximum number of nodes expanded vs. depth of goal
heuristic function K₂ for different values of W
840 to 895 algorithm executions per value of W

W	W = .5	W = .2	W = .7	W = 1.0 Breadth-first	
1	1	1	1	1	1.000
2	2	5	2	2	2.600
3	3	7	3	3	6.800
4	4	14	4	4	14.050
5	6	20	6	6	26.450
6	8	31	8	10	44.925
7	12	51	13	111	81.025
8	17	83	17	232	128.580
9	29	145	29	216	220.200
10	32	208	55	305	386.630
11	39	311	89	446	648.230
12	59	447	104	432	950.900
13	90	747	196	564	1578.300
14	93	1270	258	434	2673.200
15	146	1819	426	577	3205.400
16	299	3058	434	572	
17	307	3965	771	572	
18	380		653	488	
19	497		789	336	
20	600		1116	428	
21	951		890	369	
22	1051		921	460	

23	1622	1252	587
24	2241	1373	421
25	2794	1478	613
26	4251	1268	418

Figure 2.4-2 Estimate of distance to goal vs. actual distance
Heuristic K_1

N	KMAX(N)	KMIN(N)	KMEAN(N)
0	0	0	
1	1	1	1.000
2	2	2	2.000
3	3	3	3.000
4	4	3	3.805
5	5	3	4.684
6	6	3	5.152
7	7	4	5.717
8	8	3	5.972
9	8	3	6.239
10	8	2	6.399
11	8	1	6.569
12	8	2	6.587
13	8	3	6.650
14	8	3	6.647
15	8	3	6.675
16	8	3	6.714
17	8	4	6.777
18	8	4	6.818
19	8	5	6.733
20	8	4	6.739
21	8	4	6.588

Figure 2.4-3 Estimate of distance to goal vs. actual distance
heuristic K_2
Analogous to Figure 2.4-2

N	KMAX(N)	KMIN(N)	KMEAN(N)
0	0	0	.000
1	1	1	1.000
2	2	2	2.000
3	3	3	3.000
4	4	4	4.000
5	5	5	5.000
6	6	4	5.985
7	7	5	6.784
8	8	4	7.588
9	9	5	8.366
10	10	4	9.068
11	11	3	9.669
12	12	4	10.174
13	13	5	10.591
14	14	4	11.094
15	15	5	11.488

16	16	4	11.948
17	17	5	12.451
18	18	4	12.957
19	19	5	13.488
20	20	4	13.954
21	21	7	14.511
22	20	10	15.086
23	21	9	15.293
24	22	10	16.178
25	21	11	16.488
26	20	10	16.500

Figure 2.4-4 Estimate of distance to goal vs. actual distance
heuristic K_2
Analogous to Figure 2.4-2

N	KMIN(N)	KMAX(N)	KMEAN(N)
0	0	0	.000
1	1	10	3.668
2	2	17	7.574
3	3	18	11.060
4	4	22	14.384
5	5	32	17.650
6	6	39	20.839
7	7	40	23.692
8	8	44	26.412
9	9	51	28.727
10	10	52	31.237
11	11	56	33.378
12	12	60	35.034
13	13	58	36.582
14	14	59	38.446
15	15	60	39.824
16	16	64	41.555
17	17	62	43.254
18	18	64	43.863
19	19	60	44.963
20	20	62	45.806
21	29	60	47.178
22	33	66	48.257
23	34	64	48.973
24	35	66	50.444
25	34	62	50.000
26	36	61	50.250

Figure 2.5-2 Mean number of nodes at level l of search tree
Depth of goal = $N = 20$, $W = .5$
 K_1 and K_2 have large "mid-depth bulge" (HDB)

l	K_3	K_1	K_2
0	1.000	1.000	1.000
1	1.425	2.525	2.275
2	1.000	4.625	4.000

3	2.000	8.850	7.150
4	2.175	15.150	11.400
5	2.400	24.000	15.300
6	2.625	41.325	21.950
7	3.025	70.150	28.775
8	3.500	113.600	33.125
9	3.450	175.650	31.725
10	3.750	287.100	31.025
11	3.125	456.200	24.875
12	3.225	598.300	20.375
13	3.100	455.900	13.275
14	3.400	268.200	9.200
15	3.025	92.300	5.150
16	3.350	34.400	3.325
17	3.050	6.975	1.125
18	2.775	1.950	1.000
19	2.300	1.100	1.000
20	1.550	1.000	1.000
21	1.200		
22	1.125		
23	1.000		

Figure 2.5-3 LEVMEAN interval fraction function for data of Figure 2.5-2
 MDB = mean value of ILIF(p) - pI

p	K ₁	K ₂	K ₃
.250	.050	.100	.150
.500	.150	.250	.500
.750	.250	.350	.600
.900	.400	.550	1.000

Figure 2.5-4 Mean run length during search
 3 heuristics for 8-puzzle, W = .5
 700 to 895 algorithm executions per curve

N	K ₂	K ₁	K ₃
0	.000	.000	.000
1	1.000	1.000	1.000
2	2.000	2.000	2.000
3	3.000	3.000	3.000
4	4.000	3.810	4.000
5	4.800	4.350	4.900
6	5.775	4.400	5.838
7	6.371	4.570	6.250
8	6.751	4.500	7.454
9	6.720	3.090	7.437
10	6.988	2.420	8.291
11	5.859	1.990	8.139
12	6.123	1.680	7.200
13	4.934	1.490	6.878
14	5.129	1.470	7.636
15	4.491	1.366	6.829
16	4.472	1.310	7.222

17	3.137	1.270	4.503
18	2.946	1.260	5.969
19	2.678	1.240	4.996
20	3.439	1.230	6.358
21	2.492		5.886
22	2.529		4.959
23	2.432		3.965
24	2.409		4.462
25	2.262		4.317
26	1.352		5.248

Tabulated Data for Figures in Chapter 3

Figure 3.3-3 $\delta(i)$ = Relative error function for K_2
(Derived from $KMIN(i)$ and $KMAX(i)$ data in Figure 2.4-3)

i = actual distance	$\delta(i)$
0	.000
1	.000
2	.000
3	.000
4	.000
5	.000
6	.200
7	.167
8	.333
9	.286
10	.429
11	.571
12	.500
13	.444
14	.556
15	.500
16	.600
17	.545
18	.636
19	.583
20	.667
21	.500
22	.333
23	.400
24	.375
25	.313
26	.333

Figure 3.3-6 Comparison of the $KMIN(i)$ functions corresponding to different $\delta(i)$ functions
(with $KMAX(i) = 1$ in each case)
The $KMIN(i)$ functions have different asymptotic growth rates.

i = actual distance	$KMIN_b(i)$	$KMIN_a(i)$	$KMIN_c(i)$	$KMIN_d(i)$
0	.000	.000	.000	.000
1	1.000	.000	.000	.600
2	.970	.667	.340	1.200
3	1.390	1.500	.800	1.800
4	1.940	2.200	1.330	2.400
5	2.560	3.333	1.910	3.000
6	3.240	4.280	2.520	3.600
7	3.950	5.250	3.160	4.200
8	4.700	6.220	3.820	4.800
9	5.470	7.200	4.500	5.400
10	6.260	8.180	5.190	6.000
11	7.060		5.900	6.600
12	7.860		6.620	7.200
13	8.710		7.356	7.800

14	9.568		8.098	8.408
15	18.488	13.125	8.888	9.888
16	11.274	14.118	9.688	9.688
17	12.238	15.111	10.368	10.288
18	13.828		11.138	10.888
19	13.988		11.918	11.488
20	14.798	18.095	12.698	12.888

Figure 3.5-1 Predicted worst case number of nodes expanded for heuristic K_2 based on $KMIN(i)$ and $KMAX(i)$ data from Figure 2.4-3

N	W = .5	W = .2	W = .7	W = 1.8
1	1.888	1.888	1.888	1.888
2	2.888	2.637	2.888	2.888
3	3.888	4.274	3.888	3.888
4	4.888	6.954	4.888	4.888
5	5.637	11.341	5.637	5.637
6	8.317	23.896	8.317	998.825
7	15.498	34.852	28.872	1684.249
8	22.679	54.896	39.316	2218.472
9	34.435	85.598	123.735	2588.798
10	53.679	137.167	261.928	2887.028
11	85.181	221.585	346.347	3413.244
12	136.758	386.884	438.765	
13	168.252	444.197	482.334	
14	199.754	678.428	513.837	
15	218.998		598.255	
16	258.588			
17	282.882			

Figure 3.5-2 Predicted vs. observed number of nodes expanded in worst case for 8-puzzle heuristic K_2

$XWORST(K_2, W, N)$ is predicted (dash), $XMAX(K_2, W, N)$ is experimental (solid)
Each data point on solid curves based on up to 48 algorithm executions (a.e.)
895 a.e. total for $W = .5$ (solid curve), and 648 a.e. total for $W = .2$ (solid curve)

N	W=.5 (solid)	W=.5 (dash)	W=.2 (solid)	W=.2 (dash)
1	1	1.888	1	1.888
2	2	2.888	5	2.637
3	3	3.888	7	4.274
4	4	4.888	14	6.954
5	6	5.637	28	11.341
6	8	8.317	31	23.896
7	12	15.498	51	34.852
8	17	22.679	83	54.896
9	29	34.435	145	85.598
10	32	53.679	288	137.167
11	39	85.181	311	221.585
12	59	136.758	447	386.884
13	98	168.252	747	444.197
14	83	199.754	1278	678.428
15	146	218.998	1819	
16	299	258.588	3858	

17	387	282.082	3965.
18	388		
19	497		
20	688		
21	951		
22	1051		
23	1622		
24	2241		
25	2794		
26	4251		

Figure 3.5-3 $XWORST(K, W, N)$ and $XMAX(K, W, N)$ for K_2 , different values of W . $XWORST(K_2, W, N)$ is predicted (dash), $XMAX(K_2, W, N)$ is experimental (solid). Each data point on solid curves based on up to 40 algorithm executions (a.e.) 897 a.e. total for each of $W = .7$ (solid curve), and $W = 1.8$ (solid curve).

N	W=.7 (solid)	W=.7 (dash)	W=1.8 (solid)	W=1.8 (dash)
1	1	1.800	1	1.800
2	2	2.800	2	2.800
3	3	3.800	3	3.800
4	4	4.800	4	4.800
5	6	5.637	6	5.637
6	8	8.317	10	998.825
7	13	28.872	111	1684.249
8	17	39.316	232	2218.472
9	29	123.735	216	2588.798
10	55	261.928	385	2887.828
11	89	346.347	446	3413.244
12	104	438.765	432	
13	196	482.334	564	
14	258	513.837	434	
15	426	598.255	577	
16	434		572	
17	771		572	
18	653		488	
19	789		336	
20	1116		428	
21	998		369	
22	921		468	
23	1252		587	
24	1373		421	
25	1478		613	
26	1268		418	

Figure 3.5-4 Analogous to Figure 3.5-2, for heuristic K_1 . $XWORST(K_1, W, N)$ is predicted (dash), $XMAX(K_1, W, N)$ is experimental (solid). Each data point on solid curves based on up to 48 algorithm executions (a.e.)

N	W=.5 (solid)	W=.5 (dash)	W=.2 (solid)	W=.2 (dash)
1	1	1.800	1	1.800
2	2	2.800	2	2.637
3	3	3.800	8	5.317
4	5	4.637	10	8.784

5	7	6.274	17	16.885
6	10	18.038	27	28.648
7	14	29.785	47	47.884
8	20	41.541	80	79.386
9	34	60.785	142	130.955
10	50	80.028	224	182.525
11	72	111.531	362	
12	99	163.100	594	
13	174		960	
14	248		1572	
15	426		2448	
16	662		3945	
17	1030			
18	1361			
19	2305			
20	3385			

Figure 3.5-5 Analogous to Figure 3.5-3, for heuristic K_1
 $XWORST(K_1, W, N)$ is predicted (dash), $XMAX(K_1, W, N)$ is experimental (solid)
 Each data point on solid curves based on up to 40 algorithm executions (a.e.)

N		W=1.0 (solid)	W=1.0 (dash)	W=.7 (solid)	W=.7 (dash)
1	1		1.000	1	1.000
2	2		2.000	2	2.000
3	3		3.000	3	3.000
4	8		5.680	5	4.637
5	82		611.903	7	6.274
6	746			13	25.518
7	742			21	77.087
8	1155			29	128.656
9	1369			63	213.075
10	569			139	
11	1250			197	
12	1249			242	
13	1001			466	
14	1121			760	
15	1146			1033	
16	943			1306	
17	1237			1771	
18	2110			1649	
19	1019			2425	
20	1699			4196	
21	1539			4391	
22	990			4391	
23	1074			5003	
24	1039				
25	1493				
26	849				

Figure 3.5-6 Analogous to Figure 3.5-2, for heuristic K_3
 $XWORST(K_3, W, N)$ is predicted (dash), $XMAX(K_3, W, N)$ is experimental (solid)
 Each data point on solid curves based on up to 40 algorithm executions (a.e.)

N		W=.5 (solid)	W=.5 (dash)	W=.2 (solid)	W=.2 (dash)

1	1	1.000	1	1.000
2	2	8.181	2	3.680
3	3	39.683	4	8.087
4	4	71.186	5	15.248
5	6	155.604	8	22.429
6	8	761.828	18	34.184
7	22	1368.051	14	65.687
8	18	1738.377	14	97.189
9	26	2108.783	23	148.758
10	69		35	233.177
11	40		56	317.595
12	111		86	369.164
13	67		98	420.733
14	86		136	505.152
15	139		227	643.345
16	158		360	
17	163		421	
18	218		656	
19	244		620	
20	201		1813	
21	161		1849	
22	234		2365	
23	482		3620	
24	186		3315	
25	382			
26	89			

Figure 3.5-7 Analogous to Figure 3.5-3, for heuristic K_3
 $XWORST(K_3, W, N)$ is predicted (dash), $XMAX(K_3, W, N)$ is experimental (solid)
 Each data point on solid curves based on up to 40 algorithm executions (a.e.)

N	W=.7 (solid)	W=.7 (dash)	W=1.0 (solid)	W=1.0 (dash)
1	1	1.000	1	1.000
2	2	12.756	2	52.569
3	3	150.949	3	1044.957
4	4	289.142	4	2037.345
5	6	659.468	6	3661.884
6	8	1651.856	9	5286.423
7	30	3276.395	74	
8	10	4900.934	11	
9	49		74	
10	69		117	
11	74		114	
12	238		217	
13	80		100	
14	90		112	
15	192		138	
16	197		142	
17	186		157	
18	138		145	
19	110		236	
20	199		220	
21	128		216	
22	197		209	
23	213		171	
24	331		170	

25	193	238
26	345	169

Figure 3.5-8 $E(K, W)$ = RMS of factor of difference between $XWORST(K, W, N)$ and $XMAX(K, W, N)$ averaged over common values of N . Each data point represents up to 895 experimental observations.

W	K ₁	K ₂	K ₃
.100	1.405	1.245	3.918
.200	1.128	1.621	
.300	1.163	1.339	
.400	1.265	1.288	
.500	1.574	1.516	
.600	1.650	2.014	
.700	2.242	2.327	
.800	2.134	2.174	
.900	2.066	3.704	
1.000	2.489		

Figure 3.5-9 $E(K, W)$ = RMS factor of difference between $XWORST(K, W, N)$ and $XMAX(K, W, N)$ (Different scale of ordinate axis from Figure 3.5-8). Experimental observations (for XMAX) based on more than 26,000 algorithm executions.

W	K ₁	K ₂	K ₃
.100	1.405	1.245	3.918
.200	1.128	1.621	10.907
.300	1.163	1.339	26.133
.400	1.265	1.288	33.663
.500	1.574	1.516	49.916
.600	1.650	2.014	73.794
.700	2.242	2.327	104.984
.800	2.134	2.174	112.964
.900	2.066	3.704	191.203
1.000	2.489	8.187	

Tabulated Data for Figures in Chapter 4

Figure 4.1.1-1 $T_f(N)$ = number of pair-tests to solve N-Queens puzzle
(to find first solution)

algorithm BACKTRACK vs. Waltz-type algorithm DEEB

one algorithm execution per plotted point (solid curves)

SAS = size of assignment space

$T_{\min}(N)$ = minimum number of pair-tests for any algorithm that solves all SAPs

N	BACKTRACK	DEEB	$T_{\min}(N) = N$	$D_{\max}(N)$	SAS(N)
4	36	78	6	72	128
5	26	189	10	210	1875
6	355	635	15	450	23328
7	96	653	21	903	478598
8	2438	2101	28	1568	
9	962	1996	36	2628	
10	3172	3816	45	4050	
11	1754	3937	55	6105	
12	11755	8444	66	8712	
13	5467	8458	78	12246	
14	119888	49338	91	16562	
15	95992	42599	105	22155	
16	827932	255159	128	28808	
17	485597	178733	136	37128	

Figure 4.1.1-2 number of pair-tests (T) and number of distinct pair-tests (D)
to find first solution (T_f , D_f) and all solutions (T_a , D_a)

N-Queens, BACKTRACK; 1 algorithm execution for every plotted point

$T_f(N)$ values same as those plotted in Figure 4.1.1-1

N	$T_f(N)$	$D_f(N)$	$T_a(N)$	$D_a(N)$	$T_{\min}(N) = N$	$D_{\max}(N)$
4	36	31	42	37	6	72
5	26	26	236	148	10	210
6	355	192	1088	331	15	450
7	96	83	5345	831	21	903
8	2438	510	23378	1503	28	1568
9	962	265	136807	2810	36	2628
10	3172	568			45	4050
11	1754	427			55	6105
12	11755	1105			66	8712
13	5467	742			78	12246
14	119888	2748			91	16562
15	95992	2589			105	22155
16	827932	4726			128	28808
17	485597	4834			136	37128

Figure 4.1.1-3 Redundancy ratio $M(N) = T(N) / D(N)$

= Total number of pair-tests executed / number of distinct pair-tests executed

N-Queens, algorithms BACKTRACK, first solution and all solutions

$T(N)$ and $D(N)$ values in computation of $M(N)$ are those in Figure 4.1.1-2

N	$M_1(N)$	$M_2(N)$
4	1.161	1.135
5	1.888	1.616
6	1.848	3.845
7	1.157	6.432
8	4.780	15.553
9	3.630	52.296
10	5.585	
11	4.188	
12	10.638	
13	7.368	
14	43.338	
15	37.888	
16	175.190	
17	188.458	

Figure 4.1.2-1 $T_1(N)$: mean, max and min values over $m(N)$ samples of random candidate value ordering, compared with $T_1(N)$ for "left-to-right" c.v. ordering N-Queens, algorithm BACKTRACK, first solution
 $m(N)$ algorithm executions for each value of N ; $30 \leq m(N) \leq 180$ (see text)
 818 algorithm executions (a.e.) total

N	"left-to-right"	mean random	max random	min random	$T_{min}(N) = N(N-1)/2$
4	36	25.133	41	18.000	6
5	26	24.367	44	16.000	10
6	355	318.438	611	28.000	15
7	96	147.900	382	39.000	21
8	2438	496.340	2243	59.000	28
9	962	735.800	3114	57.000	36
10	3172	2242.600	14288	93.000	45
11	1754	4584.700	44285	141.000	55
12	11755	6694.400	48887	234.000	66
13	5467	7815.800	46847	285.000	78
14	119888	9462.400	68585	212.000	91
15	95992	12841.800	97416	231.000	105
16	827932	13186.800			128
17	485597				138

Figure 4.1.2-2 $D_1(N)$: random vs. "left-to-right" candidate value ordering N-Queens, BACKTRACK, first solution

N	"left-to-right"	random	$D_{min}(N)$	$D_{max}(N)$
4	31	22.638	6	72
5	26	23.688	10	218
6	192	148.788	15	458
7	83	85.278	21	983
8	518	179.378	28	1568
9	285	228.218	36	2628
10	568	399.898	45	4858
11	427	557.348	55	6185
12	1185	691.468	66	8712
13	742	728.688	78	12248

14	2746	861.650	91	16562
15	2589	982.300	105	22155
16	4726		120	28800
17	4834		136	37128

Figure 4.1.2-3 $M_i(N) = T_i(N) / D_i(N)$: random vs. "left-to-right" candidate value ordering
N-Queens, algorithm BACKTRACK, first solution

N	"left-to-right"	random
4	1.161	1.081
5	1.000	1.021
6	1.849	2.004
7	1.157	1.617
8	4.780	2.325
9	3.630	2.599
10	5.585	4.230
11	4.102	5.740
12	10.638	7.019
13	7.368	7.075
14	43.330	8.490
15	37.080	9.335
16	175.190	
17	100.450	

Figure 4.1.2-4 mean $T_i(N)$ compared with $\text{prod}(N) = \text{mean } T_i(N) * \text{sol}(N)$
 $\text{sol}(N)$ = number of solutions of N-queens problem.
mean $T_i(N)$ values are those in Figure 4.1.2-1

N	mean $T_i(N)$	prod(N)
4	25.133	25.133
5	24.367	146.200
6	318.430	620.860
7	147.900	3401.000
8	496.340	22831.600
9	735.800	149367.000
10	2242.600	811821.000
11	4584.700	
12	6694.400	
13	7015.000	
14	9462.400	
15	12841.000	
16	13186.000	

Figure 4.2.3-2 DEELEV(i): Backtrack to level i before invoking DEEB
N-queens, first solution, random candidate value ordering
 $T_i(N)$ = mean number of pair-tests to solve N-Queens
Same set of 810 problem instances as in Section 4.1.2.
3 * 810 = 2430 algorithm executions total

N	DEELEV(0)	DEELEV(1)	DEELEV(2)

4	70.800	31.500	31.600
5	185.300	76.500	37.100
6	538.570	347.000	272.000
7	701.000	376.000	209.000
8	1272.000	786.000	498.000
9	1985.000	1283.000	834.000
10	3346.000	2383.000	1728.000
11	5190.000	3897.000	2986.000
12	7142.000	5468.000	4235.000
13	8718.000	6568.000	4962.000
14	11316.000	8631.000	6572.000
15	14783.000	11388.000	8802.000
16	17888.000	13865.000	10671.000

Figure 4.2.3-3 Comparison of DEELEV(i) algorithms by mean $D_f(N)$
 N-Queens, first solution, random candidate value ordering
 Data from same algorithm executions as in Figure 4.2.3-2

N	DEELEV(0)	DEELEV(1)	DEELEV(2)	$D_{min}(N)$	$D_{max}(N)$
4	42.807	22.000	20.167	6.000	72.000
5	125.300	60.630	33.070	10.000	210.000
6	256.230	175.000	114.170	15.000	450.000
7	410.130	263.700	152.930	21.000	903.000
8	638.290	449.000	301.200	28.000	1568.000
9	957.090	699.000	478.900	36.000	2628.000
10	1362.900	1044.000	760.700	45.000	4050.000
11	1871.000	1471.000	1115.000	55.000	6105.000
12	2470.100	1984.000	1537.000	66.000	8712.000
13	3104.000	2531.000	2003.000	78.000	12246.000
14	3885.000	3235.000	2593.000	91.000	16562.000
15	4838.000	4055.000	3273.000	105.000	22155.000
16	5785.000	4928.000	4044.000	120.000	28800.000

Figure 4.2.3-4 Comparison of DEELEV(i) algorithms by redundancy ratio $M_f(N) = T_f(N) / D_f(N)$
 N-Queens, first solution, random candidate value ordering
 Data from same algorithm executions as in Figure 4.2.3-2

N	DEELEV(0)	DEELEV(1)	DEELEV(2)
4	1.651	1.398	1.503
5	1.481	1.266	1.123
6	2.072	1.895	2.186
7	1.714	1.433	1.301
8	1.990	1.738	1.604
9	2.070	1.820	1.692
10	2.430	2.232	2.182
11	2.714	2.553	2.515
12	2.850	2.681	2.622
13	2.793	2.566	2.426
14	2.905	2.648	2.499
15	3.022	2.782	2.642
16	3.088	2.802	2.616

Figure 4.2.3-5 DEELEV(1) for 18-queens and 12-queens for $l = 0, 1, \dots, 9$
 first solution, random candidate value ordering
 Data for $l = 0, 1$, and 2 from same algorithm executions as in Figure 4.2.3-2
 70 algorithm executions per plotted point, 700 per algorithm, 1400 total

l	$N = 12$	$N = 18$
0	7142.000	3346.000
1	5462.000	2384.000
2	4235.000	1729.000
3	3321.000	1293.000
4	2874.000	1269.000
5	3050.000	1662.000
6	4121.000	2314.000
7	6652.000	2728.000
8	8707.000	2409.000
9	8756.000	2139.000

Figure 4.2.3-6 DEELEV(1) for 18-queens and 12-queens for $l = 0, 1, \dots, 9$
 first solution, random candidate value ordering
 Data from same algorithm executions as in Figure 4.2.3-5

l	$N = 12$	$N = 18$
0	2470.000	1363.000
1	1984.000	1044.000
2	1537.000	760.000
3	1180.000	562.000
4	939.500	443.300
5	782.600	337.000
6	636.500	279.000
7	510.400	257.900
8	449.100	251.000
9	425.000	248.700

Figure 4.2.3-7 DEELEV($N/2$) vs. DEEB by mean $T_f(N)$ and mean $D_f(N)$
 $T_f(N)$ = mean number of pair-tests to solve N -Queens
 N -queens, first solution, random candidate value ordering

N	$T_f//DEEB$	$D_f//DEEB$	$T_f//DEELEV$	$D_f//DEELEV$	$N(N-1)/2$
4	70.800	42.867	31.600	20.100	10
5	185.300	125.300			15
6	538.570	256.230	312.900	81.700	21
7	701.000	410.130			28
8	1272.000	638.290	474.200	151.000	36
9	1985.000	957.000			45
10	3346.000	1362.900	1662.000	337.000	55
11	5190.000	1871.000			66
12	7142.000	2470.100	4121.000	636.000	78
13	8718.000	3104.000			91
14	11316.000	3885.000	4091.000	932.000	105
15	14703.000	4838.000			120
16	17888.000	5785.000	4242.000	1276.000	

Figure 4.3-1 Comparison of algorithm performances by mean number of pair-tests
 N-queens, first solution, random candidate value ordering
 Same sample set for each algorithm (the one in Section 4.1.2).
 mean $T_i(N)$ values for BACKTRACK are those in Figure 4.1.2-1
 818-1818 algorithm executions per algorithm, 3448 algorithm executions total

N	BACKMARK	BACKTRACK	DEEB	BACKJUMP	$N(N-1)/2$
4	23.133	25.133	78.800	25.133	
5	23.688	24.367	185.388	24.367	18
6	164.878	318.438	538.578	288.278	15
7	86.888	147.988	781.338	129.438	21
8	197.498	496.348	1272.288	448.578	28
9	254.918	735.888	1985.888	656.138	38
10	542.548	2242.688	3346.888	1957.888	45
11	873.788	4584.788	5198.888	3965.888	55
12	1181.188	6694.488	7142.888	5743.888	68
13	1859.688	7815.888	8718.888	5864.888	78
14	1242.588	9462.488	11316.888	7838.888	91
15	1513.488	12841.888	14783.888	10691.888	105
16	1521.188	13186.888	17888.888	18877.888	120
17	2158.988				138
18	3316.588				

Figure 4.3-2 Ratio of $T_i(N)$ with "left-to-right" candidate value ordering
 to mean $T_i(N)$ with random candidate value ordering
 N-Queens, first solution, random candidate value ordering
 same set of algorithm executions as those for Figure 4.3-1
 (Values for BACKJUMP = values for BACKTRACK)

N	BACKMARK	BACKTRACK	DEEB
4	1.388	1.438	1.188
5	1.188	1.867	1.828
6	1.299	1.148	1.179
7	.956	.649	.931
8	3.651	4.918	1.658
9	1.169	1.387	1.818
10	1.398	1.418	1.148
11	.544	.383	.758
12	1.678	1.756	1.182
13	.891	.779	.969
14	9.956	12.588	4.359
15	8.188	7.475	2.897
16	43.288	62.789	14.264
17	17.688		

Figure 4.3-3 Algorithm comparison by mean number of distinct pair-tests
 N-Queens, first solution, random candidate value ordering
 Same set of algorithm executions as in Figure 4.3-1

N	DEEB	BACKTRACK & BACKMARK	$D_{min}(N)$	$D_{max}(N)$

4	42.867	22.630	6	72
5	125.300	23.600	10	210
6	256.230	140.700	15	450
7	410.130	85.270	21	903
8	638.290	179.370	28	1568
9	957.090	228.210	36	2628
10	1362.900	399.090	45	4050
11	1871.000	557.340	55	6105
12	2470.100	691.460	66	8712
13	3104.000	728.630	78	12246
14	3885.000	861.650	91	16562
15	4838.000	982.300	105	22155
16	5785.000		120	28800

Figure 4.3-4 Algorithm comparison by mean redundancy ratio
 $M_r(N) = T_r(N) / D_r(N)$
 N-Queens, first solution, random candidate value ordering
 Same set of algorithm executions as in Figure 4.3-1

N	BACKMARK	BACKTRACK	DEEB
4	1.016	1.081	1.651
5	1.080	1.021	1.481
6	1.120	2.004	2.072
7	1.014	1.617	1.714
8	1.063	2.325	1.990
9	1.069	2.599	2.770
10	1.210	4.231	2.430
11	1.316	5.740	2.714
12	1.358	7.019	2.850
13	1.283	7.075	2.793
14	1.310	8.490	2.903
15	1.333	9.335	3.022
16	1.273		3.088
17	1.390		
18	1.546		

Figure 4.3-5 Growth rate of mean $T_r(N)$ using approximation mean $T_r(N) = N^C(N)$
 hence $C(N) = \log(T_r(N)) / \log(N)$
 N-Queens, algorithm BACKMARK, first solution, random candidate value ordering
 mean $T_r(N)$ values for $N < 18$ are taken from Figure 4.3-1
 50 algorithm executions per data point for $N \geq 28$, so 1310 algorithm executions total

N	C(N)
4	2.270
5	1.960
6	2.850
7	2.290
8	2.540
9	2.520
10	2.734
11	2.820
12	2.820
13	2.720

14	2.788
15	2.788
16	2.648
17	2.718
18	2.888
20	2.648
25	2.638
30	2.758
35	2.888
40	2.718
50	2.798

Figure 4.4.2-1 $L(N)$ = link percentage (degree of constraint) for N-Queens problem

N	$L(N)$
4	.444
5	.552
6	.622
7	.676
8	.714
9	.746
10	.778
11	.791
12	.808
13	.823
14	.835
15	.846
16	.856

Figure 4.4.2-2 Analogous to Figure 4.3-1, but using sample set of randomly generated SAPs having same size and degree of constraint as N-Queens SAPs
 first solution, random candidate value ordering
 50-250 algorithm executions (a.e.) per data point
 850-1100 a.e. total per algorithm; 3900 a.e. total

N	BACKTRACK	BACKMARK	BACKJUMP	DEES	$N(N-1)/2$
4	29.660	23.140	26.380	45.940	
5	88.640	53.760	71.600	289.800	10
6	197.260	92.620	141.000	427.000	15
7	156.720	79.360	104.000		21
8	415.590	143.000	268.000	1178.000	28
9	595.070	178.000	348.000	1786.000	36
10	753.000	208.000	392.000	2609.000	45
11	1305.000	263.000	477.000	3713.000	55
12	1266.000	286.000	600.000	5135.000	66
13	3281.000	395.000	587.000	6920.000	78
14	3113.000	405.000			91

Figure 4.4.2-3 Ratio of mean $T_1(N)$ for N-Queens to mean $T_1(N)$ for "Random-N-Queens" SAPs
 Data from Figures 5.3-1 and 5.4.2-2, respectively. $3440 + 3900 = 7340$ a.e. total
 Experimental data by which to distinguish "natural" SAPs from
 parametrically similar randomly generated SAPs

N	BACKTRACK	BACKMARK	BACKJUMP	DEEB
4	.847	1.000	.958	1.540
5	.275	.439	.348	.890
6	1.574	1.780	2.040	1.260
7	.944	1.100	1.240	
8	1.194	1.380	1.720	1.080
9	1.236	1.430	1.890	1.110
10	2.975	2.710	4.980	1.280
11	3.511	3.320	8.310	1.400
12	5.285	3.850	9.570	1.350
13	2.138	2.680	10.340	1.260
14	3.040	3.070		

Figure 4.4.2-5 Experimental data to distinguish "natural" problems from parametrically similar i.i.d.-random problems
Ratio of $M_1(N)$ for N-Queens to $M_1(N)$ for random N-queens
first solution, random candidate value ordering

N	BACKTRACK	BACKMARK	DEEB (dashed line)
4	.357	.999	1.180
5	.667	.955	.840
6	1.040	1.040	1.150
7	.902	.994	
8	.936	.982	1.070
9	.940	.990	1.070
10	1.560	1.130	1.190
11	1.640	1.200	1.280
12	2.130	1.270	1.250
13	1.270	1.120	1.180
14	1.730	1.180	

Figure 4.4.3-1 Dependence of mean number of pair-tests (T_p) on degree of constraint (L)
150 randomly generated SRPs of size $N = K_1 = 10$ for each plotted point
1350 a.e. per algorithm, 5400 a.e. total
upper solid curve: BACKTRACK; middle: BACKJUMP; lower: BACKMARK
first solution

L	BACKTRACK	BACKMARK	BACKJUMP	DEEB
.000	100.000	100.000	100.000	100.000
.100	223.000	186.000	189.000	378.100
.200	452.000	299.000	379.000	1162.000
.300	985.000	524.000	794.000	3776.000
.400	2546.000	1046.000	1945.000	4568.000
.500	8710.000	2682.000	6259.000	7552.000
.600	35598.000	7791.000	22551.000	16360.000
.650	15893.000	3046.000	9636.000	
.700	9143.000	751.000	2346.000	3152.000
.800	352.000	137.000	171.000	2686.000
.900	67.400	67.200	67.400	2743.000
1.000	45.000	45.000	45.000	1385.000

Figure 4.4.3-2 Dependence of mean number of distinct pair-tests (D_f) on L
 Same set of algorithm executions as in Figure 4.4.3-1
 upper solid curve: BACKTRACK and BACKMARK; lower curve: BACKJUMP
 (lower and upper solid curves have almost identical values)
 first solution

L	BACKTRACK & BACKMARK	BACKJUMP	DEEB
.000	100.000	100.000	100.000
.100	186.300	169.400	268.300
.200	292.100	272.600	548.900
.300	472.700	447.000	1193.800
.400	794.000	758.000	1994.000
.500	1426.000	1377.000	2347.000
.600	2267.000	2205.000	2826.000
.650	1138.000	1076.000	
.700	431.700	399.400	1518.000
.800	131.900	118.700	1286.000
.900	67.190	67.190	1107.000
1.000	45.000	45.000	855.000

Figure 4.4.3-3 Dependence of mean redundancy ratio (M_f) on L
 Same set of algorithm executions as in Figure 4.4.3-1
 upper solid curves: BACKTRACK; middle: BACKJUMP; lower: BACKMARK
 first solution

L	BACKTRACK	BACKMARK	BACKJUMP	DEEB
.000	1.000	1.000	1.000	1.000
.100	1.198	1.003	1.116	1.366
.200	1.541	1.026	1.386	2.106
.300	2.071	1.107	1.767	3.123
.400	3.185	1.313	2.547	2.288
.500	6.044	1.869	4.502	3.202
.600	14.410	3.180	9.411	5.532
.650	10.890	2.166	6.953	
.700	7.204	1.473	4.460	2.050
.800	1.910	1.021	1.277	2.094
.900	1.002	1.000	1.002	2.489
1.000	1.000	1.000	1.000	1.528

Figure 4.5.1-2 Mean number of pair-tests to 4-color a planar map
 first solution, random candidate value ordering
 50 algorithm executions per data point, 1000 total

N	BACKTRACK	BACKJUMP	BACKMARK
5	11.900	11.900	11.900
10	52.000	52.000	43.540
15	88.800	88.800	76.220
20	136.400	136.400	113.800
25	150.100	150.100	131.200
30	177.700	177.700	150.500
34	405.400	359.000	