A Space-Time Tradeoff for Memory-Based Heuristics

Robert C. Holte and **István T. Hernádvölgyi**
University of Ottawa
School of Information Technology & Engineering
Ottawa, Ontario, K1N 6N5, Canada
Email: {holte,istvan}@site.uottawa.ca

Abstract

A memory-based heuristic is a function, $h(s)$, stored in the form of a lookup table (pattern database): $h(s)$ is computed by mapping s to an index and then retrieving the appropriate entry in the table. (Korf 1997) conjectures for search using memory-based heuristics that $m \cdot t$ is a constant, where m is the size of the heuristic's lookup table and t is search time. In this paper we present a method for automatically generating memory-based heuristics and use this to test Korf's conjecture in a large-scale experiment. Our results confirm that there is a direct relationship between m and t.

Introduction

A heuristic is a function, $h(s)$, that computes an estimate of the distance from state s to a goal state. In a *memory-based heuristic* this computation consists of mapping s to an index which is then used to look up $h(s)$ in a table. Even heuristics that have a normal functional definition are often precomputed and stored in a lookup table in order to speed up search ((Prieditis 1993), (Korf 1997)). For other heuristics the tabular form is the most natural – for example, the pattern databases of (Culberson & Schaeffer 1996). Pattern databases are an important recent advance in heuristic search; they have been instrumental in efficiently solving very large problems such as Rubik's Cube (Korf 1997) and the 15-Puzzle (Culberson & Schaeffer 1996).

The attraction of memory-based heuristics is that they enable search time to be reduced by using more memory. Intuitively, a larger lookup table is capable of representing a more accurate heuristic thereby reducing search time. (Korf 1997) expresses this intuitive relationship quantitatively and conjectures that memory (m) and time (t) can be directly traded off, *i.e.*, that the product $m \cdot t$ is a constant. This conjecture is very important because if it is true search time can be halved simply by doubling available memory.

In this paper we test this conjecture in a large-scale experiment in which hundreds of heuristics having a wide variety of memory requirements are evaluated; in total 236,100 problem instances are solved. For this experiment a method was required to generate a wide variety of heuristics automatically. The next two sections describe our representation for state spaces and the method used to generate memory based heuristics. Subsequent sections discuss the conjecture in more detail and present our experiment's design and results.

State Space Representation

A state space is defined by a triple $S =< s_0, O, L >$, where s_0 is a state, O is a finite set of operators, and L is a finite set of labels. The state space consists of all states reachable from s_0 by any sequence of operators.

To facilitate the automatic generation of many different abstractions of widely varying granularity, we use a simple vector notation for states and operators instead of more conventional representations such as STRIPS (Fikes & Nilsson 1971). A state is represented by a fixed length vector of labels from L. An operator is represented by a left-hand side (LHS) and right-hand side (RHS), each a vector the same length as the state vectors. Each position in the LHS and RHS vectors may be a constant (a label from L), a variable, or an underscore (_). The variables in an operator's RHS must also appear in its LHS. An operator is applicable to state s if its LHS can be unified with s. The act of unification binds each variable in LHS to the label in the corresponding position in s. RHS describes the state that results from applying the operator to s. The RHS constants and variables (now bound) specify particular labels and an underscore in a RHS position indicates that the resulting state has the same value as s in that position. For example,

$$< A, A, 1, \_, B, C > \rightarrow < 2, \_, \_, \_, C, B >$$

is an operator that can be applied to any state whose first two positions have the same value and whose third position contains 1. The effect of the operator is to set the first position to 2 and exchange the labels in the last two positions; all other positions are unchanged.

We call this notation PSVN ("production system vector notation"). Although simple, it is expressive enough to specify succinctly all finite permutation groups (*e.g. Rubik's Cube*) and the common benchmark problems for heuristic search and planning (*e.g. sliding tile puzzles*).

State Space Abstraction

A *domain abstraction* is a map $\phi : L \rightarrow K$, where L and K are sets of labels and $|K| \leq |L|$. From a domain abstraction one can induce a state space abstraction, $S' = \phi(S) =< \phi(s_0), \phi(O), K >$, by applying ϕ to each position of s_0 and to every label in the LHS and RHS of each operator in O. This definition extends the notion of "pattern" in the pattern database work (Culberson & Schaeffer 1996), which in their framework is produced by mapping several of the labels in L to a special new label ("don't care") and mapping the rest of the labels to themselves.

The key property of state space abstractions is that they are homomorphisms and therefore the distance between two states in the original space, S, is always greater than or equal to the distance between the corresponding abstract states in $\phi(S)$. Thus, abstract distances are admissible heuristics for searching in S (in fact they are monotone heuristics: for formal proofs of these assertions see (Hernádvölgyi & Holte 1999)).

The heuristic defined by an abstraction can either be computed on demand, as is done in Hierarchical A* (Holte *et al.* 1996), or, if the goal state is known in advance, the abstract distance to the goal can be precomputed for all abstract states and stored in a lookup table (pattern database) indexed by abstract states. In this paper we take the latter approach.

The memory required for a pattern database, m, is clearly just its size, $m = |\phi(S)|$, and this can vary from 1 (if ϕ maps all labels to the same value) to $n = |S| =$ the number of states in S (if ϕ is one-to-one). m can even be much larger than n. This happens if the image of S under ϕ is embedded in and connected to a larger space. We call such abstractions non-surjective and in our experiments we have intentionally avoided them since Korf's conjecture (see below) is certainly false for them. The simplest example of a non-surjective homomorphism occurs with a state space defined by $s_0 =< 1, 2, 3, 4 >$ and one operator

$$op: < A, A, B, C > \rightarrow < A, A, C, B >$$

that is not applicable to s_0 because it requires the first two vector positions to be equal. S contains only one state, s_0. If $\phi(1) = \phi(2)$ and $\phi(3) \neq \phi(4)$ then $\phi(op)$ will be applicable to $\phi(s_0)$ and $\phi(S)$ will contain two states. Non-surjective abstractions do arise in practice. All our attempts to represent the Blocks World in PSVN have given rise to non-surjective homomorphisms (Hernádvölgyi & Holte 1999).

We can therefore generate a memory-based heuristic, simply by generating a domain abstraction ϕ and using it as a state space homomorphism. The memory needed for the heuristic is directly proportional to the granularity of ϕ (*i.e* K) and how many labels in L are mapped to each label in K.

Korf's Conjecture

A fundamental question about memory-based heuristics concerns the relationship between m, the size of the pattern database for a heuristic, and t, the number of nodes generated when the heuristic is used to guide search. (Korf 1997) gives an insightful, but informal, analysis of this relationship which leads to the conjecture that $t \approx n/m$. That t should be inversely proportional to m is a direct consequence of approximating t by b^{d-e} where b is the effective branching factor (the number of children of a node in the search tree), d is the average optimal solution length, and e is the average value of the heuristic being used. $b^{d-e} = b^d/b^e$ and, assuming that S and $\phi(S)$ are both tree-shaped with the same branching factor, $b^d/b^e \approx n/m$.

There are two data points of experimental evidence touching this conjecture. In (Korf 1997), the search space was Rubik's Cube and $t \cdot m$ equaled $1.4\,n$. This is in almost perfect agreement with the conjecture but it must be noted that the experiment used two independent pattern databases simultaneously whereas the analysis was based on the use of a single pattern database. In (Culberson & Schaeffer 1996) the search space was the 15-puzzle and, for the best single pattern database used, $t \cdot m$ equaled $17.3\,n$. This is in fair agreement with the conjecture , but in these experiments the Manhattan distance heuristic was used in conjunction with the pattern database and symmetries of the puzzle were used to reduce the pattern database size and increase its accuracy, so the observed performance is not attributable to the pattern database alone. Finally, both (Korf 1997) and (Culberson & Schaeffer 1996) used IDA* (Korf 1985) which produces a larger t than the search model used in the analysis.

The use of the average value of the heuristic, e, in the (Korf 1997) analysis is a weakness that is improved in (Korf & Reid 1998). There the number of nodes generated (t) in the worst case is shown to be

$$t(b, d, P) = \sum_{i=1}^{d+1} b^i P(d - i + 1) \qquad (1)$$

where b and d are as above and $P(x)$ is the probability that a state $s \in S$ has a heuristic value $h(s) \leq x$. Thus the average heuristic value is replaced by the probability distribution over the heuristic values. In a private communication, Richard Korf has suggested that equation 1 can be approximately re-expressed in terms of m

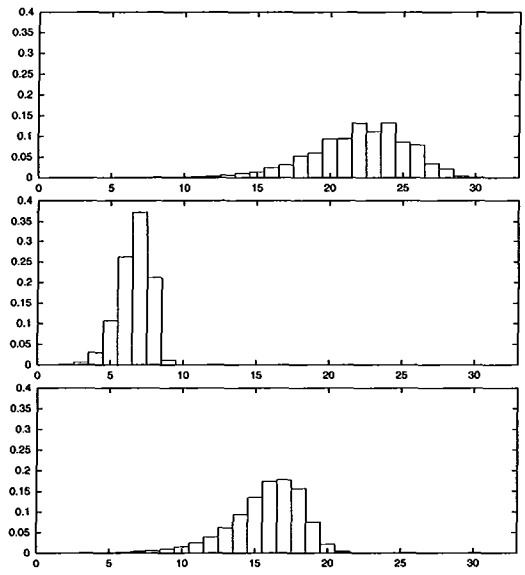

Figure 1: Solution lengths histogram for the 8-Puzzle (top), 8-Perm (middle) and (8,4)-Top-Spin (bottom).

as follows:

$$t \approx n \frac{log_b m}{m} \quad (2)$$

Experiment Design

The aim of our experiments is to examine the true relationship between t and m and compare it with the relationships conjectured in (Korf 1997) and (Korf & Reid 1998). Our approach is to create abstractions with different values of m and problem instances with different values of d and measure t by running A* (not IDA*) with each abstraction on each problem instance[1]. This is repeated for different search spaces to increase confidence in the generality of our conclusions.

	8-Puzzle	8-Perm	Top-Spin
n	181440	40320	40320
min m	252	56	28
max m	30240	20160	10080
b	1.667	6	2
d_{easy}	18	5	12
$d_{typical}$	22	7	16
d_{hard}	27	9	19

Table 1: Experiment Parameters

[1]instead of the number of nodes *generated* we measure the number of nodes *expanded* by A*; they differ by a constant so our conclusions are not affected.

For a given m there can be many different abstractions. 30 are generated at random and their t values averaged[2]. t is estimated separately for "hard", "typical", and "easy" problem instances using 100 randomly selected start states of each type (the goal state is fixed for each search space). The difficulty of a problem instance is determined by how its solution length compares to the solution lengths of all other problem instances. For example, we use the median of the solution lengths to define a "typical" problem instance. Figure 1 shows the distribution of solution lengths for all problem instances in each space.

Results are presented (figure 2) as plots with m on the x-axis and t on the y-axis. Each data point represents the average of 3000 runs (30 abstractions, each applied to 100 problem instances).[3] Breadth-first search was also run on all problem instances; it represents the extreme case when $m = 1$. In total, our experiments involved solving 236,100 problem instances.

We chose state spaces large enough to be interesting but small enough that such a large-scale experiment was feasible. Table 1 gives the general characteristics and experiment parameters for each space. Note that the m values for each space range from very small to a significant fraction of n. Each state space is generated by a puzzle, which we now briefly describe.

The 8-Puzzle is composed of 8 labeled sliding tiles arranged in a 3×3 grid. There is one tile missing, so a neighboring tile can be slid into its place. In PSVN each position in the vector corresponds to a particular grid position and the label in $vector[i]$ denotes the tile in the corresponding grid position. For example, if vector position 1 corresponds to the upper left grid position, and vector position 2 corresponds to the upper middle grid position, the operator that exchanges a tile in the upper left with an empty space (λ) in the upper middle is

$$< X, \lambda, \_, \_, \_, \_, \_, \_ > \rightarrow < \lambda, X, \_, \_, \_, \_, \_, \_ >$$

In the N-Perm puzzle a state is a vector of length N containing N distinct labels and there are $N-1$ operators, numbered 2 to N, with operator k reversing the order of the first k vector positions. We used $N = 8$. In PSVN operator 5, which reverses the first 5 positions, is represented

$$< A, B, C, D, E, \_, \_, \_ > \rightarrow < E, D, C, B, A, \_, \_, \_ >$$

The (N,K)-Top-Spin puzzle has N tokens arranged in a ring. The tokens can be shifted cyclically clockwise or counterclockwise. The ring of tokens intersects a region

[2]for the smallest values of m for each space fewer than 30 abstractions sometimes had to be used; never were fewer than 18 abstractions used for a given m.

[3]except for the smallest values of m for which fewer than 30 abstractions had to be used.

K tokens in length which can be rotated to reverse the order of the tokens currently in the region. We used $N = 8$ and $K = 4$, and three operators to define the state space

$$< I, J, K, L, M, N, O, P > \rightarrow < J, K, L, M, N, O, P, I >$$
$$< I, J, K, L, M, N, O, P > \rightarrow < P, I, J, K, L, M, N, O >$$
$$< A, B, C, D, \_, \_, \_, \_ > \rightarrow < D, C, B, A, \_, \_, \_, \_ >$$

Experimental Results

Figure 2 plots the experimental results with m on the x-axis and t on the y-axis. The scale on both axes is logarithmic but the axes are labeled with the actual m and t values. With both scales logarithmic $t \cdot m =$ constant c, the conjecture in (Korf 1997), would appear as a straight line with a slope of -1. Note that the y-axis is drawn at the smallest m value used in the experiments, not at $m = 0$.

In the top part of figure 2 a short horizontal line across each line (at around $m = 4000$) indicates the performance of the Manhattan Distance on the test problem instances. This shows that randomly generated abstractions of quite small size (5040 entries, less than 3% of the size of the state space) are as good as one of the best hand-crafted heuristics known for the 8-puzzle. The best of these randomly generated heuristics expands about 30% fewer nodes than the Manhattan distance.

A linear trend is very clear in all the curves in figure 2 The correlation between the data and the least squares regression line is 0.99 or higher in every case. However, the slope is not -1. These results therefore strongly suggest that $t \cdot m^{\alpha} =$ constant c for α between -0.57 and -0.8. α in this range means that doubling the amount of memory reduces the number of nodes expanded by less than a factor of 2.

Despite the very high correlation with a straight line, it appears that the top curves in each plot, and the middle curves to a lesser extent, are bowed up, *i.e.*, that for problem instances with long solutions the effect on t of increasing m depends on the value of m, with a greater reduction in t being achieved when m is large. The lowest curve for the 8-Perm puzzle appears to be bowed in the opposite direction, which would mean increases in m give diminishing returns for problem instances with short solutions in this space.

Further evidence that the curves flatten out as they approach $m = 1$ is obtained by extrapolating the regression lines to $m = 1$ and comparing them with value with the actual value at $m = 1$ (breadth-first search). In every case the extrapolated lines are much higher than the actual value. For example, for the 8-puzzle the extrapolated lines have values at $m = 1$ of 89,322 ($d = 18$), 514,011 ($d = 22$), and 2,191,288 ($d = 27$), while the actual numbers of nodes expanded by breadth-first search are 15,725, 64,389, and 165,039 respectively. Even the regression line for the lowest curve of the 8-Perm puzzle significantly overestimates the true value at $m = 1$.

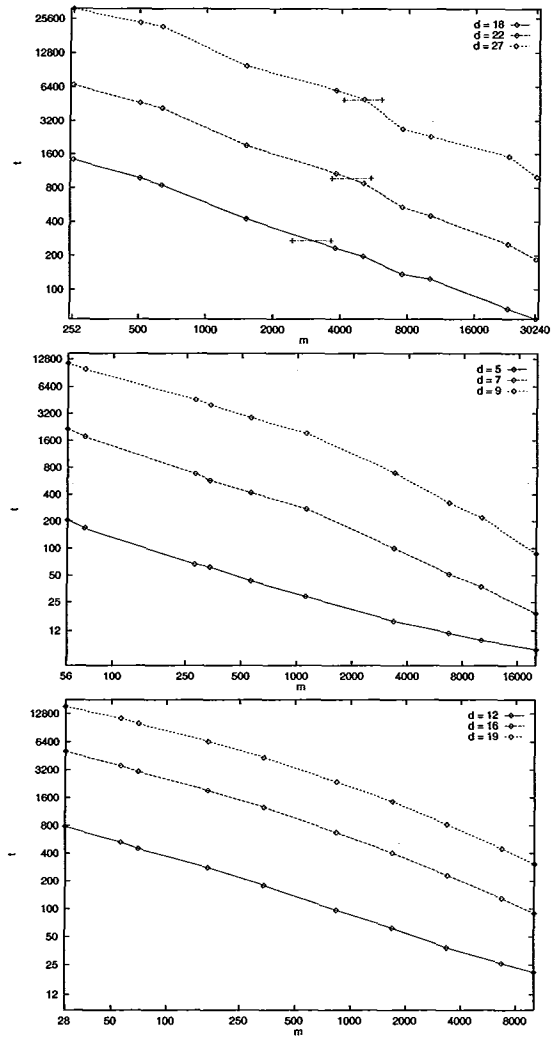

Figure 2: Number of States Expanded [t] *vs* Size of Pattern Database [m]: 8-Puzzle (top), 8-Perm (middle) and (8,4)-Top-Spin (bottom).

Similarly, the regression lines can be extrapolated to the other extreme, $m = n$, and compared to the solution length, which is the number of nodes expanded by the perfect heuristic produced when $m = n$. The lowest lines in each plot extrapolate well, giving values of 21 instead of 18 for the 8-puzzle, 4.24 instead of 5 for 8-Perm, and 8.5 instead of 12 for (8,4)-Top-Spin. The extrapolation of the middle and upper lines overestimate the value at $m = n$ significantly, the middle line's prediction being about double the true value, and the

upper line's prediction being about nine times the true value. This is yet more evidence that the middle and upper curves are not strictly linear.

The more recent conjecture is that $t \cdot \frac{m}{log_b m} = \text{constant } c$. Logarithmic scale plots of the experimental results with an x-axis of $\frac{m}{log_b m}$ instead of m produce curves that are visually indistinguishable from figure 2 but the least squares regression lines have slightly higher correlation in all cases and slopes closer to -1 (-0.67 to -0.95).

Predicting a Heuristic's Performance

One of the intuitions used in the analysis in (Korf 1997), is that the larger the average value (e) of an admissible heuristic the smaller t is expected to be. We directly tested this hypothesis for the 8-Puzzle by generating 100 random abstractions with $m = 5040$ and evaluating them on 1000 random problem instances with solution length 22 (fixed goal state, varying start state). For each of the abstractions Figure 3 shows the average heuristic value, e, on the x-axis and t, the average number of nodes expanded by A* using the heuristic on the 1000 problem instances, on the y-axis.

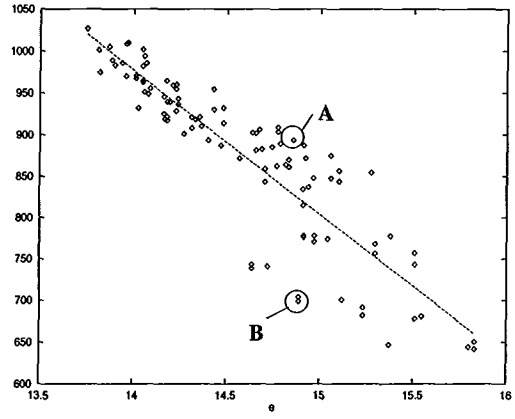

Figure 3: Number of Nodes Expanded t *vs.* Mean Heuristic Value e. 8-puzzle, $m = 5040$ and $d = 22$.

The first point of interest is the range of e. Although these heuristics all require the same amount of memory, there is a difference of more than 2 between the largest and smallest average heuristic values. The relationship between e and t is quite well represented by the least squares regression line (figure 3), except for the group of points with m values between 14.5 and 15.5 that lie well below the line. These points are extremely interesting because they plainly contradict the intuition that the larger a heuristic's average value the fewer nodes A* will expand. Consider the two heuristics A and B shown on figure 3. They both have $e = 14.71$, but A expands 873 states on average, while abstraction B expands only 703. The same trends occur with the heuristics from the previous experiment with other values of m and d and for the other search spaces.

This phenomenon is partially explained by considering P, the distribution of heuristic values in S, not only e (*as in* (Korf & Reid 1998)). It is possible to have a different P but the same average. This is illustrated by figure 4 which plots the difference between $p_A(h)$ and

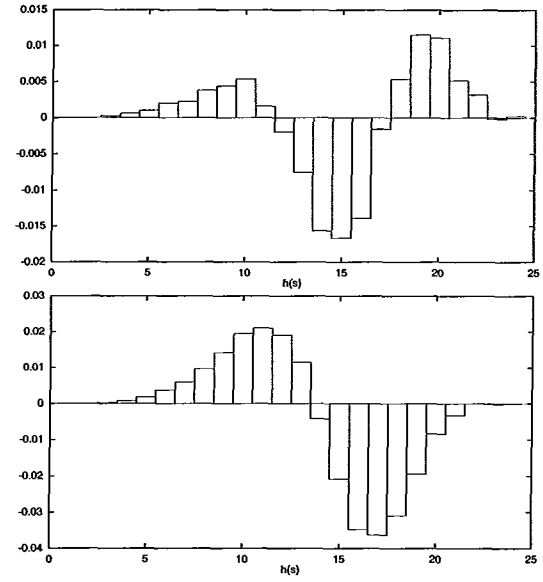

Figure 4: $p_A(h) - p_B(h)$ and $P_A(h) - P_B(h)$ *vs.* h

$p_B(h)$ *vs.* h, where h is the heuristic value provided by abstractions A and B and $p(h)$ represents the relative frequency of h within the range of heuristic values. The graph below is the cumulative difference $P_A(h) - P_B(h)$.

Although they have the same average heuristic value, abstraction B has a much higher proportion of its values near the average. By contrast abstraction A has a higher proportion of large heuristic values but also higher proportion of smaller ones. Looking at the difference in cumulative distribution, A is greater for all small values of $h(s)$. In equation 1, these values have the greatest weight, and in this case, the penalty for having more low values outweighs the savings due to having more high ones.

Conclusion

(Korf 1997) conjectured that for memory-based heuristics $t \cdot m \approx n$, where n is the size of the state space, m is the memory needed to store the heuristic as a lookup table (pattern database) and t is the number of nodes generated by A* search using the heuristic. The main result of this paper is to provide substantial experimen-

tal evidence in favor of the slightly weaker conjecture that $\log(t)$ and $\log(m)$ are linearly related. Our experiments involved using 270 different heuristics with widely varying memory requirements and solving a total of 236,100 problem instances from three different state spaces. The results had a correlation of 0.99 or greater with the least squares regression lines. There is almost certainly a small non-linear component in the relation between $log(t)$ and $log(m)$ – especially near the extremities of m's range – but for the region of most interest the relation is essentially linear. Knowing this relationship is important because it provides an assurance that any increase in memory will produce a corresponding reduction in search time.

An important feature of our experiments is that the heuristics were generated automatically – randomly, in fact. This demonstrates that the use of memory to decrease search time does not require any form of human intervention.

There are other techniques for using memory to speed up heuristic search. (Kaindl *et al.* 1995) provides a good summary and an initial comparison of some of the techniques. These techniques can be used in combination with each other and with memory-based heuristics. The optimal allocation of memory among these techniques is an open research question. Very few previous studies investigate the space-time tradeoff in detail. Perimeter search (Dillenburg & Nelson 1994) and the related BAI algorithm (Kaindl *et al.* 1995) give significant reductions in t for small values of m. For SMA* (Russell 1992), t was found to be proportional to $m^{-1.33}$ on the "perturbed 8-puzzle".

In our experiments a single memory-based heuristic was used to guide search. This was done in order to eliminate confounding factors in interpreting the results. In practice, memory-based heuristics would be used in conjunction with other knowledge of the search space. For example, (Culberson & Schaeffer 1996) uses hand-crafted memory-based heuristics in combination with the Manhattan distance and exploits symmetries in the search space and the invertibility of the operators to decrease the size and increase the usefulness of the memory-based heuristics. (Korf 1997) uses multiple memory-based heuristics simultaneously. Our experience (not reported here) indicates that $\log(t)$ and $\log(m)$ are linearly related even when multiple memory-based heuristics are used, but for a given m, t is roughly halved if three memory-based heuristics are used together instead of using just one on its own.

In this paper we have also examined the relationship between e, the average value of a heuristic, and t. For the most part our results confirm the intuition that t decreases as e increases, but the relationship is much closer to linear than the expected exponential. Furthermore, heuristics were found with relatively low e values that also had very good t values, due to a favorable distribution of heuristic values.

Acknowledgments

This research was supported in part by an operating grant and a postgraduate scholarship from the Natural Sciences and Engineering Research Council of Canada. Thanks to Jonathan Schaeffer and Joe Culberson for their encouragement and helpful comments and to Richard Korf for communicating his unpublished extensions of the (Korf & Reid 1998) work.

References

Culberson, J. C., and Schaeffer, J. 1996. Searching with pattern databases. *Advances in Artificial Intelligence (Lecture Notes in Artificial Intelligence 1081)* 402–416.

Dillenburg, J. F., and Nelson, P. C. 1994. Perimeter search. *Artificial Intelligence* 65:165–178.

Fikes, R., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2:189–208.

Hernádvölgyi, I. T., and Holte, R. C. 1999. PSVN: A vector representation for production systems. Technical Report TR-99-04, School of Information Technology and Engineering, University of Ottawa.

Holte, R. C.; Perez, M. B.; Zimmer, R. M.; and MacDonald, A. J. 1996. Hierarchical A*: Searching abstraction hierarchies efficiently. *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)* 530–535.

Kaindl, H.; Kainz, G.; Leeb, A.; and Smetana, H. 1995. How to use limited memory in heuristic search. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)* 236–242.

Korf, R. E., and Reid, M. 1998. Complexity analysis of admissible heuristic search. *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)* 305–310.

Korf, R. E. 1985. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence* 27:97–109.

Korf, R. E. 1997. Finding optimal solutions to Rubik's cube using pattern databases. *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)* 700–705.

Prieditis, A. E. 1993. Machine discovery of effective admissible heuristics. *Machine Learning* 12:117–141.

Russell, S. 1992. Efficient memory-bounded search methods. *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI-92)* 1–5.