Appeared in *Proceedings of the National Conference on Artificial Intelligence* (AAAI-98), Madison, WI, July, 1998, pp. 305–310.

Complexity Analysis of Admissible Heuristic Search

Richard E. Korf
Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095
korf@cs.ucla.edu

Michael Reid
Department of Mathematics
Brown University
Providence, RI 02912
reid@math.brown.edu

Abstract

We analyze the asymptotic time complexity of admissible heuristic search algorithms such as A*, IDA*, and depth-first branch-and-bound. Previous analyses relied on an abstract analytical model, and characterize the heuristic function in terms of its accuracy, but do not apply to real problems. In contrast, our analysis allows us to accurately predict the performance of these algorithms on problems such as the sliding-tile puzzles and Rubik's Cube. The heuristic function is characterized simply by the distribution of heuristic values in the problem space. Contrary to conventional wisdom, our analysis shows that the asymptotic heuristic branching factor is the same as the brute-force branching factor, and that the effect of a heuristic function is to reduce the effective depth of search, rather than the effective branching factor.

Introduction

We consider the asymptotic time complexity of heuristic search algorithms, such as A* (Hart et al 1968), iterative-deepening-A* (IDA*) (Korf 1985), and depth-first branch-and-bound (DFBnB), that are guaranteed to return optimal solutions. All these algorithms use the same cost function, $f(n) = g(n) + h(n)$, applied to each node n of the search space, where $g(n)$ is the cost of reaching node n from the initial state, and $h(n)$ is an estimate of the cost of reaching a goal from node n. $h(n)$ is *admissible* if it never overestimates the cost of reaching a goal from node n. These algorithms are only guaranteed to find an optimal solution, if their heuristic function is admissible (Hart et al 1968).

The time complexity of these algorithms depends primarily on the quality of the heuristic function. For example, if the heuristic returns zero for every state, these algorithms become brute-force searches, with time complexity that is exponential in the solution cost. Alternatively, if the heuristic always returns the exact cost to reach a goal, the time complexity is linear in the solution depth, assuming ties among f values are broken in favor of smaller h values (Pearl 1984). Realistic cases fall between these two extremes.

Previous Work

Most previous work on this problem (Pohl 1977, Gaschnig 1979, Pearl 1984) was based on an abstract model of the problem space and heuristic function. The model is a tree with no cycles, where every node has exactly b children. Every edge has unit cost, and there is a single goal node at depth d.

The heuristic function is characterized by its error as an estimator of actual solution cost. For example, two common assumptions are that the heuristic suffers from constant absolute error, or constant relative error. This model predicts that a heuristic with constant relative error results in linear time complexity, while constant absolute error results in exponential time complexity (Pohl 1977, Gaschnig 1979).

The main limitation of these analyses is the characterization of the heuristic function in terms of its error. In order to determine the accuracy of the heuristic on even a single state, we need to know the optimal cost to a goal from that state, which requires a great deal of computation. Doing this for a large number of states is impractical for large problems. In other words, we can't determine the accuracy of real heuristics. As a result, we cannot predict the performance of these algorithms on real problems.

Overview

This requires a different approach, which is the subject of this paper. We begin with the conditions for node expansion by any admissible search algorithm. Next, we give an alternative characterization of a heuristic, which is simply the distribution of heuristic values over the problem space. Then we specify the assumptions of our analysis. The main result is the number of node generations as a function of the distribution of the heuristic, the depth of search, and the branching factor of the problem space. Finally, we compare our analytic predictions with actual data on both Rubik's Cube and sliding-tile puzzles, using well-known heuristics. One of the implications of this analysis is that, contrary to current belief, the effect of a heuristic function is to decrease the effective depth of search, rather than the effective branching factor.

Conditions for Node Expansion

We measure asymptotic time complexity by the number of node generations, which is b times the number of node expansions, where b is the branching factor. This assumes that a node can be generated and evaluated in constant time. While some implementations of A* require logarithmic time to select the best node to expand next, and some heuristic functions require more than constant time to evaluate, these additional costs are usually polynomial, however, and depend on particular implementation choices. The dominant factor in time complexity, however, is the number of node generations, which is usually exponential in solution cost, and relatively independent of the particular admissible algorithm chosen and its implementation. The set of nodes expanded can be characterized by their cost, relative to the optimal solution cost. We begin with the conditions for node expansion by A*, then consider IDA* and depth-first branch-and-bound.

A heuristic function $h(n)$ is *consistent* if for any node n and any neighbor n', $h(n) \leq k(n,n') + h(n')$, where $k(n,n')$ is the cost of the edge from n to n' (Pearl 1984). Consistency is similar to the triangle inequality of metrics, and is usually satisfied in practice. If $h(n)$ is consistent, then $f(n) = g(n) + h(n)$ is monotonically nondecreasing along any path from the root node. Thus, the sequence of costs of nodes expanded by A* starts at the heuristic value of the start state, and stays the same or increases until it reaches the cost of an optimal solution. Some nodes with the optimal solution cost may be expanded, until a goal node is chosen for expansion, at which point the algorithm terminates. This means that all nodes n whose cost $f(n) < c$ will be expanded, where c is the optimal solution cost, and no nodes n whose cost $f(n) > c$ will be expanded. In other words, $f(n) < c$ is a sufficient condition for A* to expand node n, and $f(n) \leq c$ is a necessary condition. For a worst-case analysis, we adopt the necessary condition.

If the heuristic function is inconsistent, then the conditions for node expansion are more complex. However, most naturally occurring admissible heuristic functions are consistent (Pearl 1984). Furthermore, an inconsistent but admissible heuristic is easily transformed into a consistent admissible heuristic, which is more accurate than the original one (Mero 1984).

An easy way to understand the node expansion condition is that any admissible search algorithm must continue to expand every possible solution path until its cost is guaranteed to exceed the cost of an optimal solution, lest it lead to a better solution. Thus, while this condition was originally derived for A*, it also applies to IDA* and depth-first branch-and-bound.

On the final iteration of IDA*, the one that finds a goal, the cost threshold will equal c, and in the worst case, IDA* will expand all nodes n whose cost $f(n) \leq c$. If the number of nodes grows exponentially with cost, the previous iterations will not effect the asymptotic time complexity of IDA* (Korf, 1985).

For depth-first branch-and-bound (DFBnB), once an optimal solution is found, the upper bound on solution cost will equal c. From then on, DFBnB will expand only those nodes with $f(n) < c$. Until then, while the upper bound exceeds c, DFBnB will expand some nodes with $f(n) > c$. However, locally ordering the internal nodes of the search tree by cost often results in finding an optimal solution fairly quickly, with most of the time spent verifying that the solution is indeed optimal. Thus, we measure the asymptotic time complexity of all three algorithms by the number of nodes n whose total cost $f(n) = g(n) + h(n) \leq c$, where c is the cost of an optimal solution.

Characterization of the Heuristic

The previous analyses characterized the heuristic function in terms of its accuracy of estimating optimal costs. As mentioned above, this is very hard to determine for a real heuristic, since obtaining optimal solutions is computationally very expensive.

By contrast, we characterize a heuristic by the distribution of heuristic values over all nodes in the problem space. In other words, all we need to know is the number of nodes that have heuristic value zero, one, two, etc. Equivalently, we specify this distribution by a set of parameters $P(x)$, which is the fraction of total nodes in the problem space whose heuristic value is less than or equal to x. We refer to this set of values as the *overall distribution* of the heuristic function, assuming that every state in the problem space is equally likely. For all values of x greater than or equal to the maximum value of the heuristic, $P(x) = 1$.

The overall distribution is easily obtained for most real heuristics. For heuristics that are implemented by table-lookup, or *pattern databases* (Culberson and Schaeffer 1996), the distribution can be determined exactly from the table. Alternatively, another way to view $P(x)$ is as the probability that a state s chosen randomly and uniformly from all states in the problem space has $h(s) \leq x$. Thus, by random sampling of the problem space, we can determine the overall distribution to any desired degree of accuracy.

For heuristics that are the maximum of several different heuristics, we can compute the overall distribution of the entire heuristic from the distributions of the individual heuristics by assuming that the individual heuristic values are independent of one another. The resulting distribution will be accurate to the extent that the independence assumption is warranted.

Note that the characterization of a heuristic function in terms of its distribution is not a measure of the accuracy of the function. In particular, it says nothing about the correlation of heuristic values with actual costs, and hence doesn't require the computation of optimal solutions to any problem instances. As a result, the overall distribution is much easier to determine in practice than the accuracy of a heuristic function.

The Equilibrium Distribution

While the overall distribution is the easiest to understand, the complexity of a search algorithm depends on a potentially different distribution called the *equilibrium distribution*. The equilibrium distribution is the distribution of heuristic values at a given depth of a brute-force search, in the limit of large depth.

In some cases, such as a Rubik's Cube problem space where a 180-degree twist is a single move, the equilibrium distribution is the same as the overall distribution. The reason is that in this problem space, ignoring any special significance attached to the standard goal state, every state is equivalent to every other state, in the sense that there exists an automorphism of the problem space that maps any state to any other state.

In general, however, the equilibrium distribution may not equal the overall distribution, for example in bipartite problem-space graphs. A bipartite graph is one where the set of nodes can be divided into two subsets so that every edge goes between nodes in different subsets. For example, the problem spaces of the sliding-tile puzzles, shown in Figure 1, are bipartite. Every legal move takes the blank from an odd-numbered position to an even-numbered position. If our underlying problem space is bipartite, then the heuristic values may converge to two different equilibrium distributions, one at even levels of the tree and the other at odd levels.

	1	2
3	4	5
6	7	8

	1	2	3
5	4	7	6
8	9	10	11
13	12	15	14

Figure 1: Eight and Fifteen Puzzles

For example, the familiar Manhattan distance heuristic function for the sliding-tile puzzles is computed by measuring the distance of each tile from its goal location in grid units, and summing these values over all tiles, except the blank. This heuristic is both admissible and consistent. Since the Manhattan distance always increases or decreases by one with every move, at a given level of the search tree, the Manhattan distance of all nodes have the same even-odd parity. Thus, there are two different equilibrium heuristic distributions, one for odd values and one for even values.

Another reason for a discrepancy between the equilibrium and overall distributions is if the problem space contains different types of states. For example, in Eight Puzzle states where the distance to the goal and hence Manhattan distance is even, the blank could either be in the center or a corner location. The overall distribution assumes that any position of the blank is equally likely, but in a deep brute-force search, the blank is more likely to be in the center position than in any one of the corners (Edelkamp and Korf 1998). In this case, the equilibrium distribution of odd heuristic values is computed by combining the separate overall distributions for corner and center states, weighted by their relative asymptotic frequencies.

The equilibrium distribution is not a property of a problem, but of a problem space. For example, eliminating the parent of a node as one of its children in a problem with invertible operators will affect the equilibrium distribution. As another example, the Rubik's Cube problem space that allows only 90-degree twists as primitive operators is a bipartite graph, whereas the space that allows single 180-degree twists is not bipartite. The equilibrium distribution is defined by a brute-force search of a particular problem space, and is not affected by any pruning of the tree. Thus, it is independent of cost threshold and initial state.

In a bipartite graph, we treat even and odd levels of the search separately. For simplicity of exposition, however, we will use $P(x)$ to refer to a single equilibrium distribution. If it can't be determined exactly, it may be approximated by the overall distribution.

The Complexity Analysis

Basic Assumptions

First, we assume that our algorithm doesn't detect states that have been previously generated. Thus, multiple nodes that correspond to the same state of the problem are counted separately in our analysis. This is true of linear-space algorithms such as IDA* and DF-BnB. While A* checks for previously generated states, the resulting space complexity makes it impractical for large problems. Next, we assume that all edges have unit cost, and hence the cost of a solution is the number of edges in the solution path. We also assume that the heuristic function is integer valued. Given an admissible non-integer valued heuristic, we simply round up to the next larger integer. We also assume that the heuristic is consistent. This implies that the heuristic value of a parent is at most one greater than the heuristic value of its children.

Given these assumptions, our task is to determine $N(b, d, P)$ the asymptotic worst-case number of nodes generated by an admissible search algorithm on a tree with branching factor b, solution depth d, and a heuristic characterized by the equilibrium distribution $P(x)$. As explained above, this is the number of children of nodes n for which $f(n) = g(n) + h(n) \leq d$.

An Example Search Tree

To understand the derivation of our main result, consider Figure 2. It shows a schematic representation of a search tree generated by an iteration of IDA* to depth 8. The vertical axis represents the depth of a node below the start, and the horizontal axis represents the heuristic value. Each box represents not an individual node, but an entire set of nodes with the same depth and heuristic value, indicated by the number in the node. The arrows represent the relationship between parent and child node sets. Since the heuristic is assumed to be consistent, and furthermore, in our example problems all operators are invertible, each parent node can only generate children with heuristic values one less, one greater, or equal to that of the parent, but this latter condition is not required by our analysis. Solid boxes represent "fertile" nodes which are expanded in this iteration, while dotted boxes represent "sterile" nodes that are not expanded, because their total cost exceeds the cutoff depth. The thick diagonal line separates the fertile and sterile nodes. In this particular example, the maximum value of the heuristic is 4, and the cutoff depth d is 8 moves. We arbitrarily chose 3 for the heuristic value of the start state.

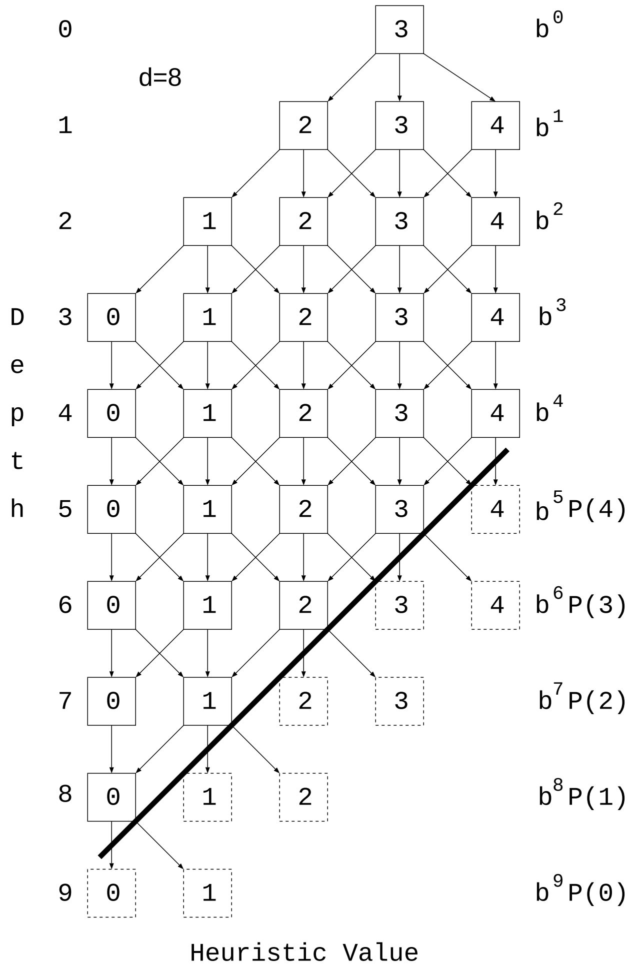

Figure 2: Sample Graph for Analysis Result

Nodes Generated as a Function of Depth

At depth 0, there is a single start state. This root node generates b children, whose heuristic values range from 2 to 4, inclusive. Each of these nodes generate b nodes, whose heuristic values will range from 1 to 4, giving a total of b^2 nodes at depth 2. Since the cutoff depth is 8, in the worst-case, all nodes n whose total cost $f(n) = g(n) + h(n) \leq 8$ will be expanded. Since 4 is the maximum heuristic value, all nodes down to depth $8 - 4 = 4$ will be expanded, and hence all nodes down to depth 5 will be generated, as in a brute-force search. Down to this depth, the number of nodes at depth d will be b^d. Note that $P(4) = 1$, and hence $b^5P(4) = b^5$, since 4 is the maximum heuristic value. In general, down to depth $d - m$, where d is the cutoff depth and m is the maximum heuristic value, all nodes are expanded, and up to depth $d - m + 1$, all nodes are generated. Asymptotically in the limit of large depth, the distribution of heuristic values will have converged to the equilibrium distribution by this point.

The total number of nodes at depth 6 is b times the number of fertile nodes at depth 5. The fertile nodes at depth 5 are those with $f(n) = g(n) + h(n) = 5 + h(n) \leq 8$, or $h(n) \leq 3$. Since the heuristic distribution at depth 5 is assumed to be the equilibrium distribution, the fraction of all the nodes at depth 5 with $h(n) \leq 3$ is $P(3)$. Since the total number of nodes at depth 5 is b^5, the number of fertile nodes at depth 5 is $b^5P(3)$, and the total number of nodes at depth 6 is $b^6P(3)$.

While there are nodes at depth 6 with all possible heuristic values, their distribution is not equal to the equilibrium distribution. In particular, the nodes with heuristic values 3 and 4 are underrepresented compared to the equilibrium distribution. The reason is that such nodes are normally generated by parents with heuristic values from 2 to 4. At depth 5, however, the nodes with heuristic value 4 are sterile, and hence their offspring are missing from depth 6, reducing the number of nodes at depth 6 with heuristic values 3 and 4.

The number of nodes at depth 6 with $h(n) \leq 2$ is completely unaffected by this pruning, however, since their parents are the nodes at depth 5 with $h(n) \leq 3$, all of which are fertile. In other words, the number of nodes at depth 6 with $h(n) \leq 2$ is exactly the same as in a brute-force search to depth 6, or $b^6P(2)$.

Similarly, the number of nodes at depth 7 is b times the number of fertile nodes at depth 6. The fertile nodes at depth 6 are those with $h(n) \leq 2$. Thus, the number of nodes at depth 7 is $b \cdot b^6P(2) = b^7P(2)$.

Due to consistency of the heuristic function, all the possible parents of fertile nodes are themselves fertile. Thus, the absolute numbers of nodes to the left of the diagonal line is exactly the same as in a brute-force search. In other words, the heuristic pruning of the tree has no effect on the fertile nodes, even though the distribution of the sterile nodes is affected. This is the key idea behind this analysis. If the heuristic were inconsistent, then the distribution of fertile nodes would change at every level where pruning occurred, making the analysis much more complex.

In general, the number of fertile nodes at depth i is

$b^i P(d-i)$, and the total number of nodes at depth i is b times the number of fertile nodes at depth $i-1$, or $bb^{i-1}P(d-(i-1))$ or $b^i P(d-i+1)$. The total number of nodes generated by the iteration in the worst case is

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

Heuristic Branching Factor

The *heuristic branching factor* is the ratio of the number of nodes generated in an iteration to depth d, compared to an iteration to depth $d-1$. One immediate consequence of our analysis is that the heuristic branching factor is the same as the brute-force branching factor b. This conflicts with results from previous analyses based on an abstract model (Pearl, 1984), which predict that the effect of a heuristic function is to reduce the heuristic branching factor, and hence the overall complexity, from $O(b^d)$ to $O(a^d)$, where $a < b$. Our analysis, however, shows that the effect of the heuristic is to reduce the effective depth of search, rather than the branching factor, from $O(b^d)$ to $O(b^{d-k})$, for some constant k.

Comparison with Experimental Data

We tested our analysis by predicting the nodes generated by IDA* on Rubik's Cube and sliding-tile puzzles, using well-known heuristics. In the above analysis, we used b^d to represent the number of nodes at depth d in a brute-force search. In our predictions below, we replaced the b^d terms in our formula by the actual numbers of nodes at level d. These numbers are computed in time linear in the depth, by expanding a set of recurrence relations governing the generation of different types of nodes (Edelkamp and Korf 1998).

Rubik's Cube

We first tried to predict results previously obtained on Rubik's Cube (Korf 1997). We use a problem space which allows 180-degree twists as single moves, we disallow two consecutive twists of the same face, and we only allow opposite faces to be twisted in succession in one order, since twists of opposite faces are independent and hence commutative. This space has a brute-force branching factor of about 13.34847. The median optimal solution depth is 18 moves.

The heuristic we used is the maximum of three different pattern databases (Culberson and Schaeffer 1996). It is admissible and consistent, with a maximum value of 11 moves, and a mean value of about 8.9 moves. We calculated the overall distribution of the individual heuristics exactly, then assumed independence of the three heuristics to calculate the overall distribution of the combined heuristic. We ignored goal states, completing the search iterations to various depths.

In Table 1, the left-most column shows the search depth, the center column gives the node generations predicted by our analysis, the next column shows the average number of nodes generated by IDA* for a single iteration to the given depth, and the last column gives the error. For depths 10 through 12 we averaged 1000 random problem instances, for depths 13 through 16 we used 100 instances, and for depth 17 we used 25 problem instances, due to computational limits.

Depth	Theoretical	Experimental	Error
10	1,510	1,501	.596%
11	20,169	20,151	.089%
12	269,229	270,396	.433%
13	3,593,800	3,564,495	.815%
14	47,971,732	47,916,699	.115%
15	640,349,193	642,403,155	.321%
16	8,547,681,506	8,599,849,255	.610%
17	114,098,463,567	114,773,120,996	.591%

Table 1: Nodes Generated by IDA* on Rubik's Cube

The theory predicts the data to within 1% in every case. The remaining error may be due to noise, or the independence assumption among the three different heuristics. The experimental heuristic branching factor between the last two levels is 13.34595.

The Eight Puzzle

We ran a similar experiment on the Eight Puzzle, using the Manhattan distance heuristic. Its maximum value is 22 moves, and its mean value is 14 moves. The Eight Puzzle contains only 181,440 solvable states, so the heuristic distributions were computed exactly. Three different distributions were used, depending on the whether the blank is in the center, a corner, or a side position. This gives us the exact equilibrium distributions. The number of nodes at a given depth of the tree depends on the initial position of the blank, and this is also taken into account. The average optimal solution length is 22 moves, and the maximum is 31 moves, assuming the goal has the blank in a corner.

Table 2 shows a comparison of the number of nodes predicted by our analysis for a given depth, to the number of nodes actually generated by a single iteration of IDA* to the same depth, ignoring any solutions encountered. For even depths, each experimental data point is the average of all 100,800 problem instances at an even depth from the goal, and for the odd depths is the average of all 80,640 problem instances at an odd depth. Since both the average numbers of node generations and the heuristic distributions are exact, the model predicts the experimental data exactly.

Fifteen Puzzle

We ran the same experiment on the Fifteen Puzzle, again using Manhattan distance. The average optimal solution length is 52.6 moves. Since this puzzle contains over ten trillion solvable states, we can't compute the heuristic distribution exactly. Rather, we used a

Depth	Theoretical	Experimental	Error
20	793	793	0.0%
21	1,490	1,490	0.0%
22	2,386	2,386	0.0%
23	4,480	4,480	0.0%
24	7,170	7,170	0.0%
25	13,442	13,442	0.0%
26	21,509	21,509	0.0%
27	40,344	40,344	0.0%
28	64,553	64,553	0.0%
29	121,020	121,020	0.0%
30	193,634	193,634	0.0%

Table 2: Nodes Generated by IDA* on Eight Puzzle

Depth	Theoretical	Experimental	Error
40	118,847	108,685	8.55%
41	253,193	234,588	7.35%
42	539,403	502,267	6.88%
43	1,149,144	1,077,126	6.27%
44	2,448,134	2,313,858	5.48%
45	5,215,496	4,936,650	5.35%
46	11,111,071	10,632,238	4.31%
47	23,670,978	22,591,563	4.56%
48	50,428,548	48,752,514	3.32%
49	107,432,751	103,255,669	3.89%
50	228,874,246	223,159,051	2.50%

Table 3: Nodes Generated by IDA* on Fifteen Puzzle

random sample of ten billion states, that were generated in a way that guaranteed they were solvable, to approximate the overall distribution. Six different distributions were used, for each combination of the blank in a middle, corner, or side position, and at odd and even depths from the goal. The mean heuristic value is about 37 moves, and the maximum is 62 moves.

Table 3 is similar to Tables 1 and 2. Each line is the average of 10,000 random solvable problem instances, whose solution depths are the same parity as the search depth. There is enormous variation in the nodes generated in individual problem instances, for depth 50 ranging from 2 nodes to over 38 billion, for example. The agreement between theory and experiment improves almost monotonically with increasing depth, as expected for an asymptotic result. At depth 50, the error between theory and experiment is within 2.5%.

Conclusions and Further Work

We presented the first analysis of the time complexity of admissible heuristic search that predicts performance on real problems. Our characterization of the heuristic is simply the distribution of heuristic values, information that is easily obtained by random sampling. We compared our analytic predictions with experimental data on Rubik's Cube, the Eight Puzzle, and the Fifteen Puzzle, getting agreement within 1% for Rubik's Cube, exact agreement for the Eight Puzzle, and than 2.5% for the Fifteen Puzzle on typical solution lengths. Contrary to previous results, our analysis and experiments indicate that the asymptotic heuristic branching factor is the same as the brute-force branching factor, and hence the effect of a heuristic is to reduce the effective depth of search, rather than the effective branching factor.

We presented an asymptotic analysis for a fixed-size problem as the solution length grows large. The asymptotic results provide excellent predictions at typical solution depths. Ideally, we would like to predict the performance of a fixed heuristic as the problem size increases. For example, what is the asymptotic performance of the Manhattan distance heuristic on sliding-tile puzzles, as the puzzle grows large? This remains an open problem for future research.

Acknowledgments

We would like to thank Eli Gafni, Elias Koutsoupias, and Mitchell Tsai for several helpful discussions. R. Korf is supported by NSF grant IRI-9619447.

References

Culberson, J.C., and J. Schaeffer, Searching with pattern databases, in *Advances in Artificial Intelligence*, Gordon McCalla (Ed.), Springer Verlag, 1996.

Edelkamp, S. and R.E. Korf, The branching factor of regular search spaces, *Proceedings of the National Conference on Artificial Intelligence (AAAI-98)*, Madison, WI, July, 1998.

Gaschnig, J. *Performance measurement and analysis of certain search algorithms*, Ph.D. thesis. Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., 1979.

Hart, P.E., N.J. Nilsson, and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions on Systems Science and Cybernetics*, Vol. SSC-4, No. 2, July 1968, pp. 100-107.

Korf, R.E., Depth-first iterative-deepening: An optimal admissible tree search, *Artificial Intelligence*, Vol. 27, No. 1, 1985, pp. 97-109.

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

Mero, L., A heuristic search algorithm with modifiable estimate, *Artificial Intelligence*, Vol. 23, 1984, pp. 13-27.

Pearl, J. *Heuristics*, Addison-Wesley, Reading, MA, 1984.

Pohl, I., Practical and theoretical considerations in heuristic search algorithms, *Machine Intelligence 8*, 1977, pp. 55-72