EXACTLY HOW GOOD ARE HEURISTICS?: TOWARD A REALISTIC PREDICTIVE THEORY OF BEST-FIRST SEARCH[1]

John Gaschnig
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa. 15213

Abstract

We seek here to determine the exact quantitative dependence of performance of best-first search (i.e., A* algorithm) on the amount of error in the heuristic function's estimates of distance to the goal. Comparative performance measurements for three families of heuristics for the 8-puzzle suggest general conjectures that may also hold for more complex best-first search systems. As an example, the conjectures are applied to the coding pnase of the PSI program synthesis system. A new worst case cost analysis of uniform trees reveals an exceedingly simple general formula relating cost to relative error. The analytic model is realistic enough to permit reasonably accurate performance predictions for an 8-puzzle heuristic. The analytic results also sharpen the distinction between "Knowledge itself" and the "Knowledge engine itself".

> 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?
>
> J. Bronowski, The Ascent of Man

Introduction

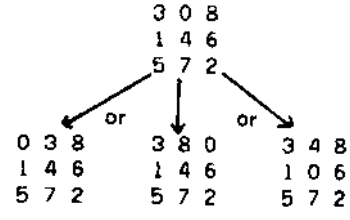
Building speech understanding systems or other expert problem solving systems can be liKened to medieval cathedral building: it can be done but it is by no means easy to do so. This paper attempts to show how some performance measurement experiments with the 8-puzzle and some mathematics of tree search can hope to ease the burden a-little, as civil engineering has done for cathedral builders.

Some of the difficulties in building such systems arise from an inability to predict performance a priori, to "calculate the stresses", so to speak. Suppose it's design decision time for such a system: Which will give better performance, heuristic A or heuristic B? Debate is sometimes avoided by using both in a multi-term evaluation function, if this system uses an evaluation function of some sort to guide behavior. But performance depends on how much weight each term is given. The choice of the scalar weight values remains largely a matter of educated gnoses combined with trial and error [^.g., Hayes-Roth & Lesser 77]. These prediction questions -- which is better? what weighting value? — constitute the focus of the experiments and the analysis reported belcw.

It is a certainty that in the future we will understand more of AI more mathematically than today. But if we somehow succeed in developing an exact mathematical model of the HEARSAY-II system [Hayes-Roth & Lesser 77], say, complete with formulas to predict performance quantitatively under the most diverse parameter settings, will such a theory say anything at all about the HWIM speech understanding system [Woods 76], or about the F 51 program synthesis system [Barstow 77, Barstow & Kant 77], or about a chemistry synthesis program [Powers 75]? This will be possible only if the theory captures a common denominator of these systems. The four systems mentioned use best-first schedulers to decide what to do next.

The A* algorithm [Hart et.al. 68, Nilsson 71] embodies the idea of a best-first search. In basic terms, you have a finite set of discrete options of what to do next, and each time you choose an action and do it, you get 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 *(<md* remember that some don't exist until you perform others), then one approach is to assign a number to each action as it appears, according to how good you think it is to do that one, independently of any you may have done already. Then iteratively choose the action that has the smallest value (smallest is best). The A* algorithm [Hart et.al. 68, Nilsson 71] operates on this principle, using an arbitrary ordering function P(s) to solve problems like the 8-puzzle. The Graph Traverser and the HPA algorithm are essentially the same as A* [Doran & Michie 66, Pohl 69].

The 8-puzzle [Schofield 67] is a one-person game the objective of which is to rearrange a given configuration of 8 tiles on a 3x3 board into another given configuration by iteratively sliding a tile into the orthogonally adjacent empty location, like so:



This problem can be modeled exactly as a collection

This research was supported by the Defense Advanced Research Projects Agency under Contract no. F44620-73-C-0074 and monitored by the Air Force Office of Scientific Research.

of points (configurations) and lines connecting them (moves), i.e., as a graph. Since some problems differ only in that they are defined by different graphs, we define a problem to be a finite, sjnongly connected graph with no self- loops and no multiple edges. (Throughout, underlining indicates a formal definition.) Actually, the 8-puzzle graph is not strongly connected, but rather consists of two disconnected components ("can't get there from here"); for our purposes we consider one such component. The 8-puzzle is an undirected graph since every move has an inverse.

The 8-puzzle is to this work as the fruit fly is to the geneticist: simple, convenient to manipulate, yet exhibiting interesting phenomena that hypothetically hold for a broader class of subjects. To approach a predictive theory of heuristic search experimentally, we define search performance functions (e.g., number of nodes expanded and length of solution path found), measure their values experimentally over a range of realistic parameter values (e.g., as a function of N, the distance to the goal, and of W, a weighting coefficient in the evaluation function), look for patterns in the data (e.g., cost grows (sub-)exponentially with N under certain conditions), and conjecture certain general relations to hold (e.g., increasing W changes an exponential cost heuristic into sub-exponential).

Cost, Quality and Error for 8-puzzle Heuristics

Heuristic search is supposed to be better than breadth-first search, but how much better? Do heuristics boat the "exponential explosion" that besets breadth-first search? Figure 1 shows the number of nodes expanded as a function of the depth of the goal for three particular heuristics for the 8-puzzle. The qualitative difference between Kg and the other two is of particular interest: could this have been predicted a priori? This work differs from previous experiments [Doran & Michie 66, Michie 67, Doran 68, Michie & Ross 70] in: a) volume of data collected, giving statistical significance over a large range of parameter values; b) measures of the error in the heuristics themselves; c) different measures of internal behavior. A few definitions are required to make Figure 1 meaningful. (Note: what we call K, [Hart et.al. 68] calllv)

Fach possible choice of initial node s_r and goal node s of a problem graph defines a distinct problem instance *[$,iJipl** hence a graph G having V nodes induces a set U(G) of V^ problem instances. The minimum distance m *he_ £l_[a]P.h between any two nodes sj and S: is always defined since the graph is strongly connected, and is denoted

This section assumes that the A* evaluation function takes the form F(s) « R(S) + K(s), where g(s) is the distance of node s from the root node of the search tree. Note that K is a function of two nodes of G (i.e., current and goal), but for simplicity we write K(s) instead of K(s, s_p) when goal node *s„* is implicit. A K function estirrates the distance in the graph from s to the goal node; informally, K contains the knowledge or information about the graph G available to guide the search.

For a given G, K, and (s_r, $s_„$), the cost of search and the goodness of the solution found can be defined, respectively by: X(G. K. s_r. s,), the number of nodes expanded before search terminates, excluding the goal nodr; and P(G, K, s_r, s.), the length pi the solution path foundKG. K. s_r Sg.) ->(G, K, $s_{r>}$ s_r) / h(s_{rt} s_c) expresses solution quality as a fraction of *\he_* minimal len^tti cM a "•9![ll]U°J2 path for an instance. So L > 1, with equality iff a minimal length solution is found. (We will conveniently drop arguments from formulas when the argument is known implicitly.)

We will consider three K functions for the 8-puzzle taken from the literature [Doran & Michie 66, Nilsson 71].

Kj(s) = the number of tiles that occupy a board location in s different from the location occupied by that tile in the goal node [c] 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_p, 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^ up to rotation about the board perimeter, and counts 2 for each tile not followed by the same tile as in the goal node.

For comparison purposes we also measure the performance of KQ(S) * 0, which gives breadth-first search.

What can be predicted about the performances of these three K functions for arbitrary problem instances? Little, beyond that L - 1 for Kj and K_2 (by the A* admissibility theorem [Hart et.al. 68], since Kj(c.j, s.) *s* h(r.j, s:) for all (s_{jt} s.) and similarly for K_2). Regarding the X measure, formal theory [Hart et.al. 68, Pohl 69, Pohl 70, Nilsson 71, Harris 74, Vanderbrug 76] tells us nothing about these particular heuristics for this particular problem, not even that X(K_2) is always less than X(K1). Intuitively, K3 may seem to be better than K_2 [Nilsson 71, p. 66], but is this true always, sometimes, or never? Two example problem instances suffice to suggest that it is risky to guess on the basis of limited data (example on left from [Doran & Michie 66]):

	X	p		x	P
	92	18		16	11
K*	23	18	K*	90	21

(s , s) =

(216.So8.753, 123.804.765) (485.163.702, 368.405.172)

Figure 1 shows the result of measuring X (by executing the search) for a set of 875 randomly chosen problem instances of the 8-puzzle (of 10^8 possible instances). The instances are grouped on the abscissa according to the actual minimum distance h(s_r, s_g) between initial node and goal node. For example, for N * 10 there are random 40 problem instances such that h(s_r, Sg) = 10, and hence 40 measurements of values of X(K_2, s_r, s_g). The mean of these 40 experimentally measured values is plotted as XMEAN(K_2, 10). In general, the true value of XMEAN(G, K, N) is defined to be the mean number of nodes oxpanded usinR heuristic K over al[(s ^ ^ l in U(G) such that h(s_r, s_g) »» N. The vertical bar measures twice the standard deviation of the sample XMEAN — a statistical measure of how accurately the experimentally measured value of XMEAN approximates the true value of XMEAN.

The lower dashed line in figure 1 shows perfect search: XMEAN(N) = N, if only solution path nodes are expanded. The same set of random problem instances is used for K_1, K_2, and K_3, in this and all subsequent experiments. Details and additional experimental results are given in [Gaschnig 77b].

In the above experiments, K_3 always found minimal length solution paths (i.e., L = 1) for $N \leq 9$, with $LMEAN(K_3, N) \leq 1.3$ for larger N. (LMEAN is defined in terms of L as XMEAN is defined in terms of X.) For human subjects, $1.1 \leq LMEAN \leq 3.3$ has been reported [Hayes et.al. 65, Doran & Michie 66].

The differences in performance among the heuristic functions result from the differences in the values they compute, but what is the dependency? What in fact are the values? Figures 2, 3, and 4 plot the range in a heuristic's estimates of distance to the goal vs. the actual distance to the goal. KMIN(G, K, i) is defined to be the minimum value of $K(s_j, s_g)$ over all node pairs (s_j, s_g) in U(G) such that $h(s_j, s_g) = i$. For example, $KMIN(K_2, 10) = 4 \leq K_2(s) \leq KMAX(K_2, 10) = 10$ for all s whose distance to the goal is 10. Each figure represents 11,448 distinct observations of $K(s_j, s_g)$ vs. $h(s_j, s_g)$, recorded during the same experiments as for figure 1: the s_g nodes are the goal nodes in the sample set and the s_j are the nodes along the solution path.

Behavior Measures

It is not obvious exactly how XMEAN and LMEAN depend on KMIN and KMAX, but the clear superiority of K_3 is compelling: what is A* doing differently in this case? The following experiments (Figures 5-7) show interesting relations between external performance and two measures of behavior during search: LEV(G, K, s_r, s_g, i), the number of nodes occurring at level i in the final search tree; and RUN(G, K, s_r, s_g), 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. RUN = N for optimal search (no mistakes), and equals about 1 for breadth-first search.

Figure 5 shows $LEVMEAN(K_j, N, i)$ for j = 1, 2, 3, and a representative case N = 20. 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. (A portion of the K_3 curve is cut off the bottom of this semi-log plot.) Observe for K_1 and K_2 that LEVMEAN(i) increases (and then decreases) approximately exponentially with i. In contrast, LEVMEAN(i) for K_3 is distributed more uniformly with i, suggesting that uniform distribution of LEVMEAN(i) is correlated with sub-exponential XMEAN.

We refer to exponentially distributed LEVMEAN(i) as "mid-depth bulge", quantified 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 6 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 from K_3. A possible scalar measure of mid-depth bulge (MDB) is the mean value of IF(p) - p. For the four values plotted for each K function in Figure 6 we have $MDB(K_1) = (.25 - .05 + .5 - .15 + .75 - .25 + .9 - .4) / 4 = .39$ and $MDB(K_2) = .29$ and $MDB(K_3) = .06$ Pending further experiment and mathematical analysis, we tentatively conjecture that XMEAN(G, K, N) is subexponential in N iff MDB(K) < .15.

Figure 7 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 by the MDB measure K_3 differs qualitatively from K_1 and K_2, by RUNMEAN the difference is only one of degree, suggesting no credible means of distinguishing subexponential from exponential cost heuristics. Is the similarity in form of the three curves coincidental? And why this particular three phase "decay" form? 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 be better than best-first search for large N for "poor" heuristics.

Effects of Changing Term Weight

How does performance change if instead of F(s) = g(s) + K(s) we use F(s) = K(s) or F(s) = (1 - W) · g(s) + W · K(s), where $0 \leq W \leq 1$? Certain formal analyses [Pohl 69, Pohl 70, Nilsson 71, Vanderbrug 76] suggest the value of the g(s) term for "insurance", but the results do not strictly apply to these 8-puzzle heuristics (see "Analysis" section).

Figures 8 through 11 show how performance varies with W, for W = 0.0, 0.1, ..., 1.0. Note that by definition W = .5 gives the same behavior as F(s) = g(s) + K(s), and that W = 0 gives breadth first search. Note in Figure 8 that for K_2, as W increases the functional form of XMEAN becomes subexponential in N, and that for medium-sized values of N, $XMEAN(K_2, N, W)$ increases as W increases. The same holds for K_1 and K_3 (not shown), except that K_3 has "reached its limit" at W = .5 (no further improvement). Figure 9 is comparable to Figure 1, but uses W = 1.0 instead of W = .5. Note in Figures 1 and 9 that for every N, K_3 is better than K_2 is better than K_1 (with statistically insignificant exception); the same is true for each value of W measured. Hence determining which heuristic is best for small N (cheap experiment) tells which is best for large N too. This may not be true generally. Figure 10 compares the three heuristics by length of the solution path (LMEAN) for W = 1.0. Note that K_3 is now better than K_2 and K_1, whereas for W = .5 the opposite is true. 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 quality by changing W but not by changing K. Figure 11 shows this cost/quality tradeoff explicitly as W varies from 0.0 to 1.0, for a "mid-sized" case N = 15: increasing W

beyond a certain value brings poorer quality at greater cost.

Implications for the PSI System

Will comparable experiments with more complex best-first search systems yield results similar to those reported here? To be concrete, we consider one such system, the program construction phase of the PSI program synthesis system, which is being implemented as a best-first search [Barstow 77, Green 77]. We interpret the current experimental results as if they applied to some extent to PSI.

The coding subsystem of PSI converts a high level program specification into a legal LISP implementation by applying rules to refine a specification into a slightly more detailed specification. The rule set induces a tree in which the terminal nodes correspond to legal programs for the given input. These target programs can differ drastically in efficiency, so that a 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 (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) guides the search in an attempt to keep both L and X acceptably small [Barstow & Kant 76]. The variable N, the number of refinement steps, might refer to the length of the shortest terminating path, or to the length of the path to the most efficient program.

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 $F(s) = (1 - W) \cdot g(s) + W \cdot K(s)$.

1) For W = .5, unless the efficiency expert (EE) is very good, it will not be feasible to synthesize 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 1)

2) By simply choosing W = 1.0 instead of W = .5, XMEAN(N) becomes sub-exponential: it will cost somewhat more to synthesize medium-sized programs, but far less to synthesize large programs (Figures 8, 9). However, the synthesized programs may be less efficient than if W = .5 is used (Figures 10, 11).

3) If EE is improved so as to reduce XMEAN, then the improvement will be observed for every value of W (Figures 1, 9; compare K_1 to K_2, or K_2 to K_3). Furthermore, for large W, the improvement in speed will also cause an improvement in the efficiency of the synthesized programs: faster heuristics find better solutions (Figure 10).

4) The XMEAN performance of a version of EE for large N can be predicted by measuring mid-depth bulge for medium-sized N (Figures 5, 6).

5) The program will hop around the search tree quite a lot unless N is small (Figure 7). Mean run length < 2 indicates poor EE. In this case, ordered depth-first search may be better than best-first search.

Worst Case Cost Analysis

The objectives of analysis of A* are both practical and theoretical. A practical objective is to find a formula for XMEAN(G, K, W, N), say, valid for an arbitrary problem graph G, heuristic function K, and over the range of W and N. Then to determine for a particular G_0 whether heuristic A performs better than heuristic B, simply evaluate the formula with $G = G_0$ once with $K = K_A$ and once with $K = K_B$; to find a good value of W, evaluate for different values of W. This most general objective may not be feasible, but Figure 12 illustrates an application of the general results described in this section: a reasonably accurate prediction of the worst case performance of K_2 for the 8-puzzle.

A principal theoretical objective of A* analysis is to determine the exact dependence of performance on error in the heuristic estimates. For a broad range of heuristic functions it turns out that worst case cost (number of nodes expanded) is a simple exponential function of a function $\delta(i)$ that measures the relative error in the heuristic estimates, as a function of distance from the goal, thus:

XWORST(M, KMIN, KMAX, N) "rel"

$$\sum_{1 \le i \le N} \lfloor M^{i \cdot \delta(KMIN, KMAX, i)} \rfloor \qquad (1)$$

where "rel" denotes "≤" if $\delta(i)$ is weakly monotonic decreasing for $i > 0$, and "≥" if $\delta(i)$ is weakly monotonic increasing for $i > 0$. The remainder of this section describes the above results in somewhat more detail. Full detail and proofs and other results are given in [Gaschnig 77a].

Until we discover how to plug an arbitrary problem graph and heuristic function into a mathematical formula (as English text? as ALGOL source code?), we must resort to making simple models that capture only a part of what we want, but which hopefully have some limited predictive power. To simplify, here we model an arbitrary graph by a single positive integer M, the branching factor of a uniform tree. A single node at level N of the tree is distinguished as the goal node. A K function on the uniform tree becomes simply the KMIN(i) and KMAX(i) functions that bound the estimates of that K function. So the heuristic is now represented by a pair of real-valued functions on the non-negative integers (e.g., Figures 2, 3, and 4). 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 any longer, but can only give the best case or average case or worst case performance. The latter can be defined as

XWORST(M, KMIN, KMAX, W, N),

which is a particular function of the form

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

where $\mathbb{N}$ denotes the non-negative integers and $\mathbb{R}$ the non-negative reals. XWORST gives the number of nodes expanded by A* in searching the uniform tree, using the following K function, for a given KMIN and KMAX:

$$KWORST(s) = \begin{cases} KMAX(h(s)) & \text{if node } s \text{ is on the solution path} \\ KMIN(h(s)) & \text{if } s \text{ is not on the solution path} \end{cases}$$

Note that KWORST returns relatively high values when it should give low values, and vice versa.

The above is an informal description of a standard formal worst case cost model [Pohl 69, Pohl 70, Nilsson 71]. Heretofore, however, XWORST results have concentrated on KMIN and KMAX functions that take the form KMIN(i) = i - a and KMAX(i) = i + b, where a and b are real-valued constants, i.e., straight lines with slope 1. So really the KMIN and KMAX functions in such analyses can be represented by real numbers a and b. This simplifies the analysis, but excludes heuristics whose KMIN and KMAX are more arbitrary, e.g., those in Figures 2, 3, and 4.

To get the general case, we note first that the objective is to determine the dependence of performance on error. The KMIN and KMAX functions, however, do not measure error. They bound the numerical estimates computed by a K function. To denote the relative error, or spread, in these estimates, we can rewrite

$$KMIN(i) = (1 - \delta(i)) \cdot \beta(i)$$

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

and solve for $\delta(i)$ and $\beta(i)$ in terms of KMIN(i) and KMAX(i):

$$\delta(KMIN, KMAX, i) = \frac{KMAX(i) - KMIN(i)}{KMAX(i) + KMIN(i)}$$

By this definition, $\beta(i)$ is the arithmetic mean of KMIN(i) and KMAX(i), and $\delta(i)$ is the relative error of KMIN(i) or KMAX(i) with respect to $\beta(i)$, for example Figure 13. By definition, $0 \le \delta(i) \le 1$ for any KMIN and KMAX.

Formula (1) holds for all "well-behaved, never overestimating" K functions. The formal conditions are that W = .5; that KMAX(i) = i, as is true of K_2 for i ≤ 21 (Figure 3). Also, that $0 \le KMIN(i) \le KMAX(i)$ and that for all i, $KMIN(i + 1) \ge KMIN(i) - 1$, as is true for K_1, K_2, and K_3 for all i plotted in Figures 2, 3, and 4. Note that $\delta(i)$ = constant is weakly monotonic both increasing and decreasing, hence formula (1) is exact (i.e., equality holds). In this case, KMIN(i) is a straight line through the origin.

Formula (1) tells how much error in a K function can be tolerated and still achieve a given level of performance. For example, for each of the KMIN functions shown in Figure 14, the corresponding $\delta(i)$ is plugged into formula (1) and simplified, to obtain the following results:

$\delta(KMIN, KMAX, i)$	XWORST(M, KMIN, KMAX, N)	
c	$O(M^{cN})$	(exponential in N)
$1/\sqrt{i}$	$\le O(N M^{\sqrt{N}})$	(≤ subexponential)
$\log i / i$	$\le O(N^{\log M})$	(≤ polynomial)
c / i	$\le O(N \cdot M^{c})$	(≤ linear)

This table expresses a guarantee: if a K function meets the specified condition, then its performance is no worse than that indicated above.

Figure 15 illustrates schematically the distinction between the "Knowledge itself" (i.e., a lattice of heuristic functions) and the "Knowledge engine itself" (i.e., XWORST is a function on the lattice). Each point in the lattice represents the δ function corresponding to particular KMIN and KMAX functions. The set of all these δ functions is partially ordered under the relation $\delta_1 \le_L \delta_2$ iff $\delta_1(i) \le \delta_2(i)$ for all i = 1, 2, ... Associated with each δ function is its corresponding XWORST value (depicted by a double arrow), which is itself a function of M and N. Formula (1) determines (or at least bounds) the "lengths" of the double arrows.

For the conditions of (1), KMAX is fixed and hence specifying KMIN determines both δ and β. In the unrestricted case, 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, however, is not a linear ordering of integers or reals, but rather an infinite continuous lattice of functions on the non-negative integers.

Formula (1) only bounds XWORST, but another more complicated expression gives XWORST exactly for all KMIN and KMAX without restriction. It is used to generate the XWORST predictions, shown in Figure 12, of the performance of K_2 for the 8-puzzle, given the KMIN and KMAX values shown in Figure 3. In figure 12 XMAX is defined like XMEAN, but takes the maximum instead of the mean of the experimental values. For the 8-puzzle M was estimated to be 1.715 experimentally. Note that this prediction is a demonstration of technical feasibility, not of "commercial marketability".

The extent of agreement between predicted and measured values in Figure 12 is not particularly good in absolute terms, but is in some sense astounding, since the graph of the 8-puzzle is not a uniform tree, and the K_2 function for the 8-puzzle does not have worst case behavior like that of KWORST.

Conclusions and Future work

The experimental results show that some heuristics beat the exponential explosion and others do not. But the latter can be made to do so simply by giving more weight (W) to the distance-to-goal term (K(s)) in the evaluation function and less weight to the distance-from-root term (g(s)). However, this reduction in the number of nodes expanded (XMEAN(N)) occurs only if distance to the goal (N) is large. For medium-sized N, XMEAN actually increases with W. Why? There is a limit to what adjusting W can do: subexponential heuristics, e.g., K_3, do not get any better by increasing W. Also, if K_i has smaller XMEAN(N) than K_j for one value of W, then the same ordering holds for any other value of W. Increasing W also increases the lengths of the solution paths found: for large W, faster heuristics find better solutions, whereas for smaller W the opposite can be true. How much of this is true for PSI or other complex best-first search systems? Clearly much remains to be investigated. If predicability can be exploited in practice, then perhaps our current heuristic search techniques are only "model T"s.

The new analytic results permit predictive statements for heuristic functions that occur in practice, but much remains for future work. When, if ever, is relative worst case performance a good predictor of relative average case performance? Are there effective ways to experimentally or analytically estimate the KMIN and KMAX of a given heuristic?

Theoretically, the fact that performance is a function of relative error rather than of absolute error would seem to merit careful thought. What makes relative error more special? The "limits to growth" tabulated in the preceedling section are sobering: A* must be given a very accurate heuristic in order to guarantee good performance \n the worst case. What about average case?

The symbolic result--a mapping from a lattice of heuristic "error" functions to a lattice of performance functions-- can serve as a definition for the (worst case) performance capability of the A* algorithm. The lattice representing heuristic functions exists independently of A*; it happens to be the domain of a particular function we have called XWORST. In words, the "Knowledge itself" is distinct from the "Knowledge engine itself", for "Knowledge" in this special sense. Hence a similar XWORST function can perhaps be derived, assuming as the "Knowledge engine" ordered depth-first search or the B* algorithm [Berliner 76] instead of A*. If you give an engine more Knowledge, i.e., less errorful knowledge, then it performs better. But some engines can do more than others (or do it faster) with the Knowledge they *are* given.

I greatfully acKnowledge many fruitful discussions with Herbert Simon regarding this worK.

References

1. Rarstow, D., "A Knowledge-Based System for Automatic Program Construction", Proc. Intl. Join! Conf. Artificial Intelligence 77, 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.
8. Berliner, H., "The B* Tree Searching Algorithm: Best-First Search Combined with Branch and Bound", unpublished memo, Dept. of Computer Science, Carnegie-Mellon Univ., April 1976.
4. Doran, J., "New Developments of the Graph Traverser", in Machine Intelligence ?, E. Dale and D. Michie (eds.), American Elsevier Publ. Co., New YorK 1968.
5>. Doran, J., and D. Michie, "Experiments with the Graph Traverser Program", Proc. Royal Society of London, Series A, Vol. 294, pp. 235-259, 1966.
6. Gaschnig, J., "The Worst Case Cost of A* as a Function on a Lattice of Heuristic Error Functions", Dept. of Computer Science Technical Report, Carnegie-Mellon Univ., July 1977. (1977a)
7. Gaschnig, J., Heuristic Search Performance and its Relation to Problem "Structure", Ph. D. Thesis, Dept. of , Computer Science, Carnegie-Mellon Univ., forthcoming 1977. (1977b)
8. Green, C, Private communication. A best-first search mechanism for the program construction phase is being implemented. Other search reduction techniques will be used in conjunction with best-first search. April 1977.
9. Harris, L., "Heuristic Search Under Conditions of Error", Artificial Intelligence, Vol. 5, No. 3, pp. 217-234, North Holland Publishing Co., Amsterdam, 1974.
10. 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.
11. 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.
12. Hayes-Roth, F., and V. Lesser, "Focus of Attention in the Hearsay-II Speech Understanding System", Dept. of Computer Science Technical Report, Carnegie-Mellon Univ., January 1 977.
.13. Michie, D., "Strategy Building with the Graph Traverser", in Machine Intelligence 1, N. Qollins and D. Michie (eds.), American Elsevier Publ. Co., New York 1967.
14. 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.
15. Nilsson, N., Problem-Solving Methods in Artificial Intelligence, McGraw-Hill Book Co, New York 1971.
16. 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 1969.
1 7. Pohl, I., "Heuristic Search Viewed as Path-Finding in a Graph," Artificial Intelligence, Vol. 1, 1970.
18. Powers, G, et. al., "Optimal Strategies for the Chemical and Enzymatic Synthesis of Bihelical Deoxyrybonucleic Acids", J. American Chemical Soc, Vol. 97, p. 875, 1975.
19. 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.
?0. Vanderbrug, G., "Problem Representations and Formal Properties of Heuristic Search", Information Sciences, vol. 1 1, no. 4, 1976.
21. 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.

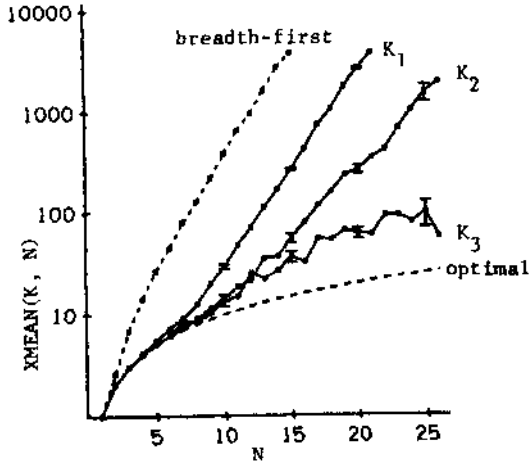


Figure 1. Number of nodes expanded vs. depth of goal for three 8-puzzle heuristics For optimal search, XMEAN(N) ■ N

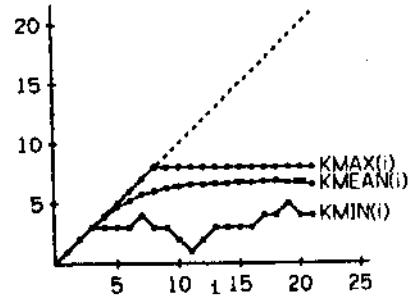


Figure 2. Heuristic estimate of distance to goal vs. actual distance, for heuristic K_1

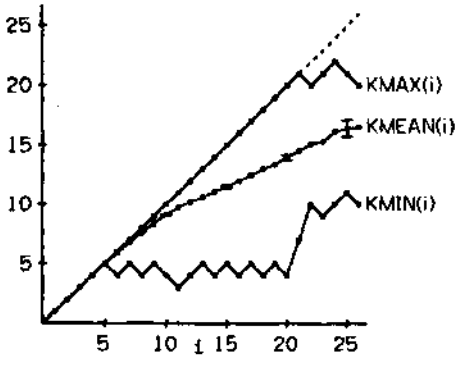


Figure 3. Same as figure 2, for heuristic K_2

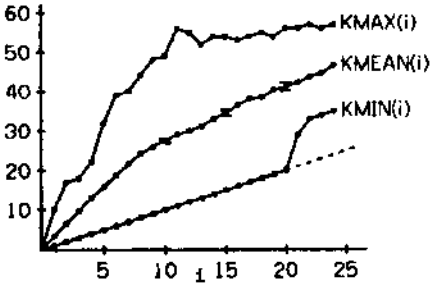


Figure 4. Same as figure 2, for heuristic K_3

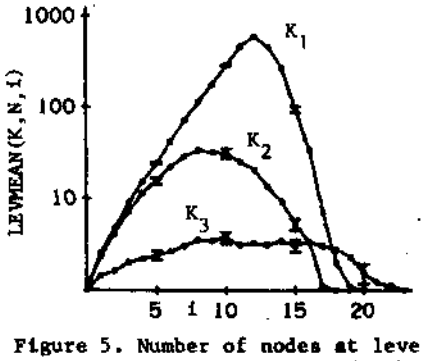


Figure 5. Number of nodes at level i in search tree; N = depth of goal = 20

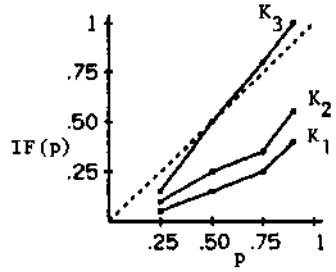


Figure 6. Interval fraction function for data in figure 5

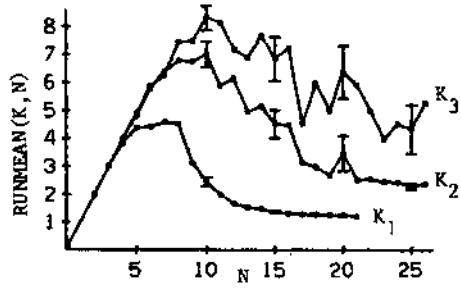


Figure 7. Mean run length vs. depth of goal

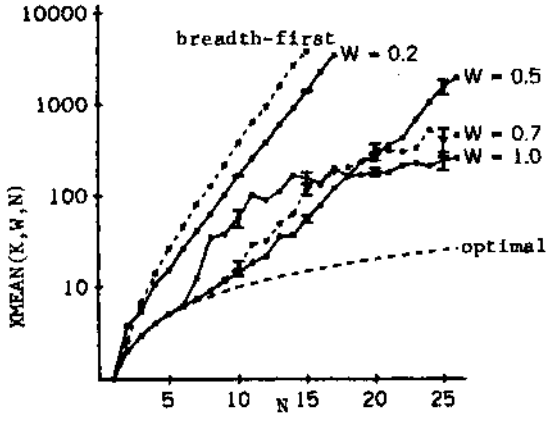


Figure 8. Number of nodes expanded vs. depth of goal; K_2 with different weight values

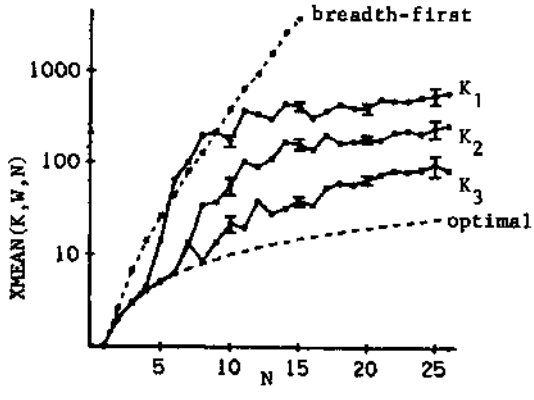


Figure 9. Number of nodes expanded vs. depth of goal, for 3 heuristics and W = 1.0 Compare with Figure 1.

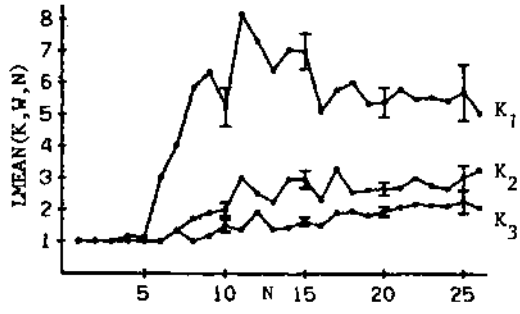


Figure 10. Solution path length ratio vs. depth of goal, at W = 1.0

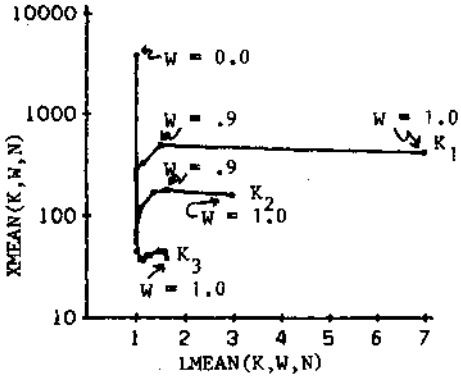


Figure 11. Cost vs. quality trade-off for fixed N = 15 as W varies

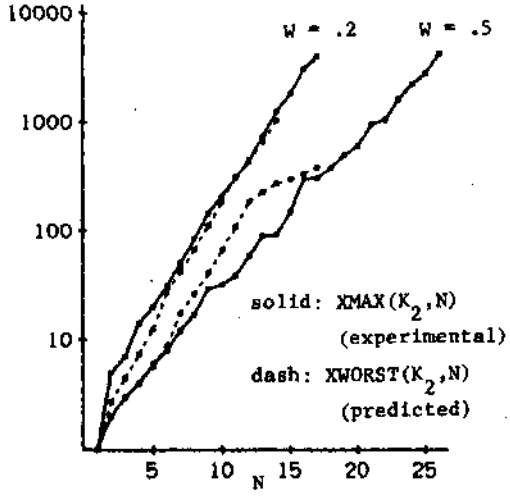


Figure 12. Predicted and measured values of worst case cost for heuristic K_2

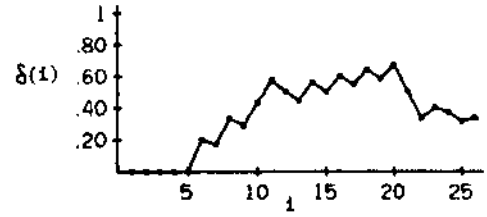


Figure 13. Relative error function for K_2 (from data in figure 3)

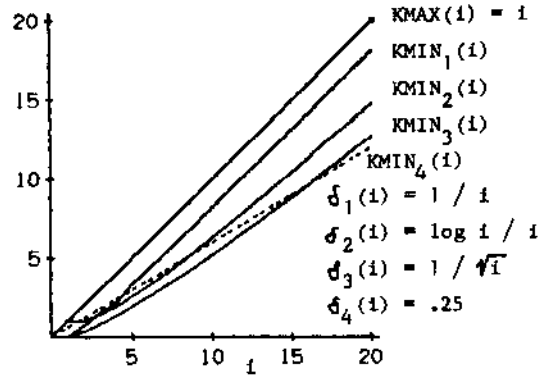


Figure 14. Fixed KMAX(i) and different KMIN(i) The KMIN functions have different assymptotic growth rates.

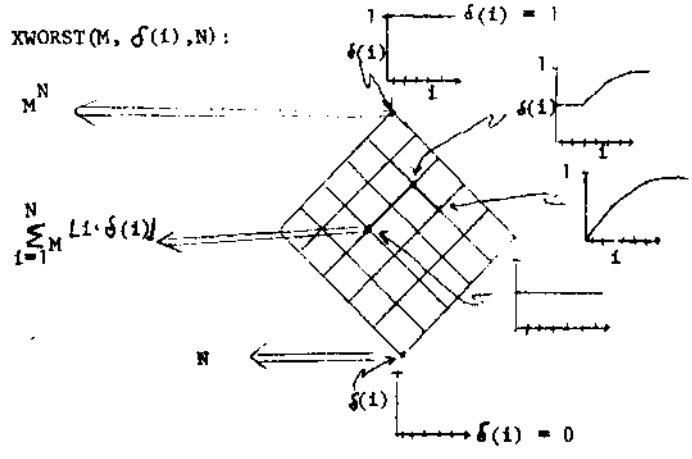


Figure 15. Finite depiction of infinite continuous lattice of heuristic functions and worst case cost as a function on the lattice. δ(KMIN,KMAX,i) is the relative error in a heuristic with bounding functions KMIN(i) and KMAX(i).