Learning Admissible Heuristics while Solving Problems

Anna Bramanti-Gregor and Henry W. Davis
Department of Computer Science
Wright State University, Dayton Ohio 45435

Abstract

A method is presented that causes *A** to return high quality solutions while solving a set of problems using a non-admissible heuristic. The heuristic guiding the search changes as new information is learned during the search, and it converges to an admissible heuristic which 'contains the insight' of the original non-admissible one. After a finite number of problems, *A** returns only optimal solutions.

Experiments on sliding tile problems suggest that learning occurs very fast. Beginning with hundreds of randomly generated problems and an overestimating heuristic, the system learned sufficiently fast that only the first problem was solved non-optimally. As an application we show how one may construct heuristics for finding high quality solutions at lower cost than those returned by *A** using available admissible heuristics.

1 Introduction

A problem with *A** is that it often gives low quality solutions when its heuristic overestimates[1]. Optimal or near-optimal solutions are often desired and a strong underestimating heuristic is not always available. In [Davis et *al.,* 1989] we describe how an admissible heuristic h_M can be derived from a non-admissible heuristic *h.* The potential savings in node expansions when *A** uses *g+hw* as an evaluator (denoted $A^*(h_M)$) is shown to be considerable when compared to previously suggested methods that attempt to find optimal solutions with non-admissible heuristics.

In this paper we extend the definition of h_M to an admissible heuristic which includes the 'collective insight' of one or more available heuristics. Using the h_M concept, we describe a method whereby, while solving a set of problems, the quality of the solutions returned by *A** can be made to steadily improve. As more problems are solved, a dynamically changing approximation to h_M, denoted Th_M, is learned. As it is learned, Th_M is also used to guide the search. We prove that, in a probabilistic sense, Th_M converges to h_{Ml} causing $A^*(Th_M)$ to be admissible after a finite number of problems have been solved.

Variations of the learning technique are described.

[1] A solution has 'high quality' when the ratio of its length to the optimal solution length is close to one.

These are chosen based on the amount of computation per problem the user wishes to invest in improving solution quality. In *constant learning* the overhead per problem is very low: a single relatively small computation. A user who is continuously using *A** in some domain may keep this type of learning active indefinitely. His system will evolve, at low cost, but slowly, towards the finding of optimal solutions. *Quadratic learning* requires more computation per problem than constant learning but Th_M converges to h_M after solving fewer problems.

In many applications high quality satisficing solutions are more desirable than optimal ones if they can be found with low time-overhead. We propose using quadratic learning to address this problem, by turning off the learning early so that the heuristic developed can be used before it evolves any further. This technique of turning off the learning early in the problem session is called here *early learning.* The purpose is to create a heuristic which (1) causes *A** to find high quality solutions, and (2) causes *A** to expand a relatively small number of nodes.

Experiments were performed in the 8-puzzle domain to get some empirical insight into the technique's effectiveness. We tested two available non-admissible heuristics, sometimes in combination with an admissible one. We used quadratic learning on samples of *1998* and *605* randomly generated problems. The results of two groups of experiments are described in this paper. In one group we gauged the speed of learning. It was fast: The preponderance of information learned is acquired within eight randomly chosen problems. In all but the first problem, during which much of the learning occurs, *A** always returned optimal solutions.

In the second group of experiments we attempted to measure the effectiveness of using the early learning method to build heuristics. We combined non-admissible heuristics with an admissible one; our goal was to learn a composite heuristic which would cause *A** to return high-quality solutions while expanding fewer nodes than when using the admissible heuristic alone. In one of the experiments reported below, the learned heuristic returned solution quality within 6% of optimal while reducing by half the number of nodes expanded by *A*.* In another experiment the system learned a heuristic which always returned optimal solutions at a 15% reduction in node expansion; however we cannot guarantee that the solutions returned will always be optimal. We conclude

that the proposed method of building composite heuristics through learning shows promise of being useful.

2 Notation and Preliminaries

We use the following notation:

R^+	Non-negative real numbers.
G	Locally finite directed graph with arc length bounded below by a positive number.
problem	An element $(s,t) \epsilon GxG$. s is viewed as the start and t as the goal.
$k(m,n)$	Length in G of minimum length path from m to n; ∞ if there is none.
$h^*(n)$	$k(n,t)$ where the goal t is implicit.
$H(m,n)$	heuristic estimate of $k(m,n)$. Assume $H(m,n) \epsilon R^+$ and $H(n,n) = 0$ for all $n \epsilon G$.
$h^t(n)$, $h(n)$	$H(n,t)$; when t is implicit we write $h(n)$. We think of t as a goal.
$g(n)$	Shortest distance yet found by A^* from start state to n.
$A^*(h^t)$, $A^*(h)$	A^* algorithm using $g + h^t$ as an evaluation function; when t is implicit, we write $A^*(h)$.
X	Number of nodes expanded by A^* on some problem.
SQ	Solution quality: ratio of length of solution returned by A^* to the optimal length.
admissible	Either $h \le h^*$ or, equivalently, $H \le k$. An algorithm is *admissible* if it returns optimal solutions when given solvable problems.

An example used later is the *8*-puzzle, a sliding tile problem (see, *eg.*, [Pearl, 1984] or [Nilsson, 1980]). A common admissible heuristic for it is the 'Manhattan distance': $H(m,n)$ denotes the sum over all tiles of the vertical and horizontal distance of each tile in m from its position in n. The problem of finding an optimal path connecting two *8*-puzzle states is part of a family of *NP*-hard problems [Ratner *et al.*, 1986].

If H is a heuristic, define for all $x \epsilon R^+$

(2.1) $MAXH(x) = \max\{H(m,n): (m,n) \epsilon GxG, k(m,n) \le x\}$.

For example, a simple enhancement of the Manhattan distance is obtained by adding to it a 'sequence score' term (see [Nilsson, 1980, p. 85]). Denote the enhancement H_3. $MAXH_3$ is shown in Figure 2.1. In general, $MAXH$ is a non-decreasing function. If G is finite, then it is a step function whose value at any jump-point is the higher of the two alternatives.[2]

h_M is defined in [Davis et al., 1989] and redefined here. Let H be a heuristic estimate (admissible or not) and let $h^t(n) = H(n,t)$. For all $n \epsilon G$ define

(2.2) $h^t_M(n) = \min\{x: h^t(n) \le MAXH(x), x \epsilon R^+\}$.

We write $h_M(n)$, instead of $h^t_M(n)$, when context allows. Definition (2.2) and the reason why h^t_M is admissible are best understood with a picture: In Figure 2.2 we let H, and its associated h, be the enhanced Manhattan distance, discussed above. Suppose, for example, $h^t(n) = 35$. (2.2) says that, to find $h^t_M(n)$ on the x axis, project horizontally to $MAXH$ from 35 on the y-axis and then vertically to the x-axis. The resulting value, 4, is defined to be $h^t_M(n)$. $h^t_M(n) \le h^*(n)$ because, if $h^*(n)$ were less than 4, say x_1, then there would be an $x_1 < 4$ with $MAXH(x_1) \ge 35$, contrary to the way we obtain $h^t_M(n)$. Thus h^t_M is admissible.

A note on the case when the H we start with is admissible: In this event $MAXH$ never rises above the diagonal of Figure 2.2, whence, $h^t_M \ge h^t$; *ie.*, h_M is at least as informed as h. This suggests a superiority of h_M over h because more informed admissible heuristics cause less node expansion [Nilsson, 1980]. However, usually $h_M = h$ because it is common for $MAXH(x)$ to lie on the diagonal when h is admissible. (For example, $h_M = h$ when h is the Manhattan distance.)

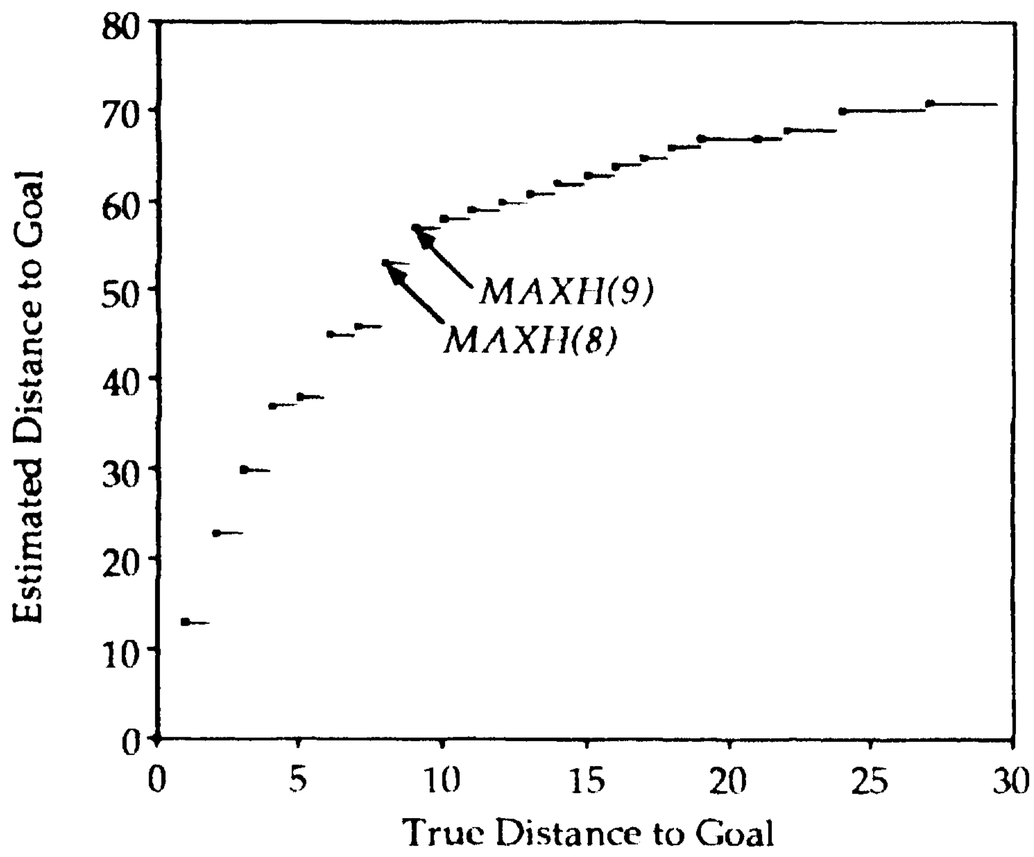

Figure 2.1. $MAXH_3$ where H_3 is the enhanced Manhattan distance. Its value at jump points is the highest of the two alternatives.

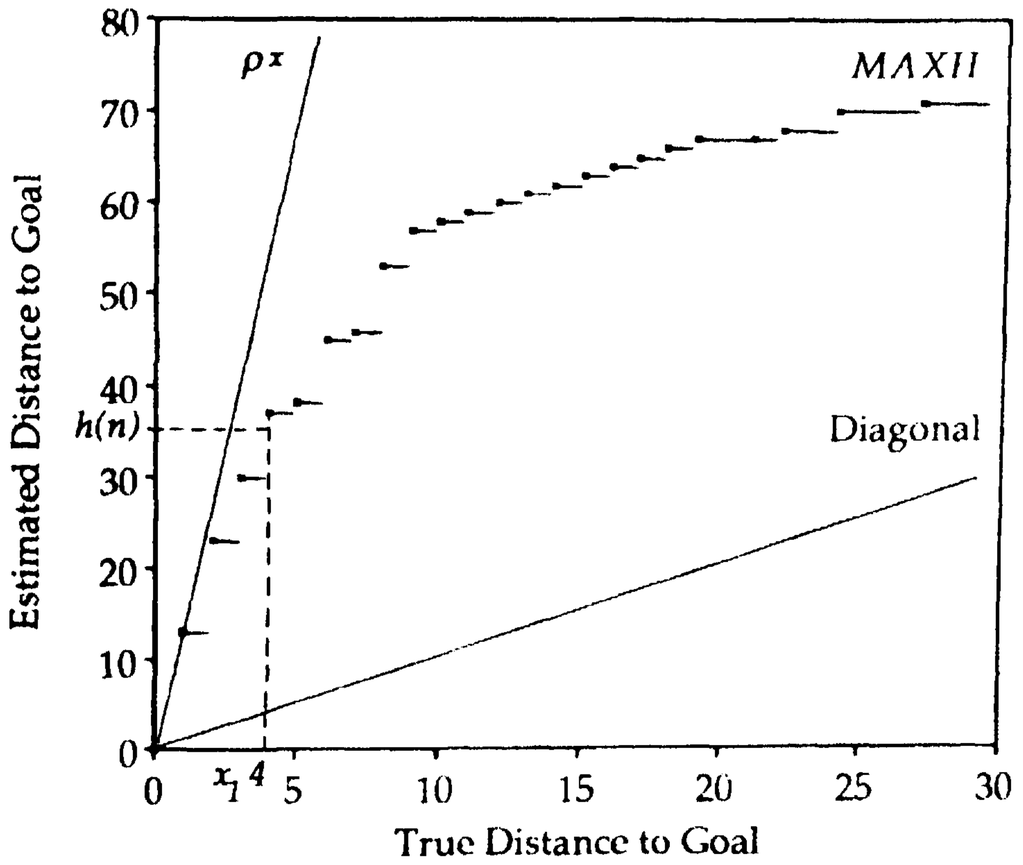

Figure 2.2. Interpretation of (2.2). See text for discussion.

[2] If G is infinite, then the 'max' operator in (2.1) and the 'min' operator in (2.2), below, should be thought of as 'supremum' and 'infimum', respectively.

Finally, we wish to extend definition (2.2) to an admissible heuristic which 'captures the insight' of several given heuristic $h_1,...,h_p$ (admissible or not). Define a heuristic $h_{(1,...,p)M}$, composed from $h_1,...,h_p$, as follows: Let h^t_{iM} be given by (2.2) and let

(2.3) $h^t_{(1,...,p)M}(n) = \max\{h^t_{1M}(n),..., h^t_{pM}(n)\}$.

$h^t_{(1,...,p)M}$ is admissible and is at least as informed as h^t_{iM}.

3 Learning and its Convergence

The critical question one asks when using h_M in (2.2) is 'how do we compute $MAXH$?' In the learning technique, statistics are collected as A^* runs enabling an approximation to $MAXH$, called $\Gamma MAXH$, to be built. This enables A^* to use an approximation to h_M, called Γh_M. Γh_M evolves over time. We discuss methods for learning $\Gamma MAXH$ at different speeds, and then extend these ideas to the case when h_M is composed from several heuristics. It is shown that, in a probabilistic sense, $A^*(\Gamma h_M)$ is admissible after solving a finite number of problems.

3.1 Quadratic learning.

Let $S = (P_1, P_2, ...)$ be a set of problems to solve. A^* is run on P_1 using a dynamic heuristic Γh_M, where h is a given heuristic. Γh_M is dynamic in the sense that a parameter used in its computation changes as more nodes are expanded. This key parameter is an estimation of $MAXH$, denoted $\Gamma MAXH$. Whenever A^* places a node n on Open, n is assigned the heuristic value $\Gamma h^t_M(n)$ where

$$(3.1)\ \Gamma h^t_M(n) = \begin{cases} h^t(n) \text{ if } h^t(n) > \Gamma MAXH(x) \text{ for all } x \in R^+ \\ \min(x: h^t(n) \le \Gamma MAXH(x),\ x \in R^+), \text{ otherwise.} \end{cases}$$

(We write Γh_M instead of Γh^t_M when context allows.) $\Gamma MAXH$ is a non-decreasing function initialized to the zero function. It is updated as follows: suppose node n has just been selected from Open by A^* and let $s = n_0, n_1, ..., n_{j+1} = n$ be the search tree path from s to n. Execute

(3.2) *For* $i = 0, 1, ..., j$ DO

$d_i \leftarrow g(n) - g(n_i)$

$\Gamma MAXH(d_i) \leftarrow max(\Gamma MAXH(d_i), H(n_i, n))$

Maintain $\Gamma MAXH$ as a non-decreasing function

The inner loop of (3.2) causes a possible increase in the recorded value of $\Gamma MAXH(d_i)$. This may cause an increase in the recorded values of $\Gamma MAXH(y)$ for many $y > d_i$ because $\Gamma MAXH$ is to be maintained as non-decreasing.

As A^* runs, (3.2) causes $\Gamma MAXH$ to grow larger through a process of statistical sampling. When it is large enough to be useful, the lower part of (3.1) applies and Γh_M approximates h_M. When not enough has been learned to use $\Gamma MAXH$, Γh_M behaves like h. Even though $\Gamma MAXH$ alters over time, we do not reassign new heuristic values to previously generated nodes. When P_1 is solved, the same procedure is now used to solve P_2, except that $\Gamma MAXH$ is initialized to the function developed during execution of P_1, rather than to the zero function. The process of solving problems in S continues in this way.

Several aspects of (3.2) should be noted. It is possible that $d_i > k(n_i,n)$ because the search tree path to n may not be optimal. Also $H(n_i,n)$ may not equal its maximum value for nodes separated by optimal paths of length $k(n_i,n)$. Both of these events contribute to make

(3.3) $\Gamma MAXH(x) \le MAXH(x)$ for all $x \in R^+$

This, in turn, causes

(3.4) $\Gamma h^t_M(n) \ge h^t_M(n)$ for all $n, t \in G$,

when $\Gamma MAXH$ has grown sufficiently large that the lower part of (3.1) is applicable.

The idea of the technique is that, as more is learned during problem solving, $\Gamma MAXH$ increases to $MAXH$, Γh_M decreases to h_M, and $A^*(\Gamma h_M)$ returns solutions of increasingly better quality while Γh_M converges to an admissible heuristic. The number of nodes expanded by $A^*(\Gamma h_M)$ on the other hand increases when compared with those expanded by A* using the unacceptable heuristic h. (This is because admissible heuristics, such as h_M, force more breadth into the search.) One could turn off this learning process when the time-quality tradeoff is acceptable, and hold the current $\Gamma MAXH$ fixed. The turn-off point is determined by pragmatics because we cannot say with certainty how close the algorithm is to convergence at any time.[3]

Let X be the number of nodes expanded by $A^*(\Gamma h_M)$ in some problem. We call the procedure of (3.2) *quadratic learning* because the amount of time per problem directly spent in learning $\Gamma MAXH$ is $O(X^2)$. To see this, notice that (3.2) is executed $X + 1$ times and, at each execution, $\Gamma MAXH$ is updated at most X times (*ie.*, $j \le X$). The space needs of learning are small: an array whose size approximates the diameter of G to hold $\Gamma MAXH$ when arc lengths are unit.

3.2 Reducing the time overhead of learning.

Linear learning differs from quadratic learning in that it takes fewer statistics per problem to learn $\Gamma MAXH$: When a node n is selected from Open we do not sample the search tree path to n, as in (3.2), but instead take a single sample:

(3.5) $\Gamma MAXH(g(n)) \leftarrow \max(\Gamma MAXH(g(n), H(s,n))$

maintain $\Gamma MAXH$ as a non-decreasing function

This is called 'linear' because the direct overhead of learning is $O(X)$ per problem. It has the virtue that it may be kept active over a long period with substantially less time overhead than in the quadratic case.

The advantage of linear learning is accentuated by *constant learning*. In this only one statistic per problem is taken while learning $\Gamma MAXH$: Namely, (3.5) is performed only if n is a goal. The direct time overhead of learning on each problem is now a constant which is independent of the problem size. This kind of learning may be kept active indefinitely with only a very low time overhead.[4]

3.3 Learning a composite of several heuristics.

Suppose, for example, we are given h_1, h_2 and we wish

[3] Several experiments with this 'early learning' technique are described in Section 4.3.

[4] In practice, constant learning should be preceded by several runs using quadratic learning. Otherwise it may be a long time before $\Gamma MAXH(x)$ is non-zero for low values of x. Learning credible values for these low x at an early time greatly enhances performance.

to form the approximation $\hat{rh}_{(1,2)M}$ which should converge to $h_{(1,2)M}$. A direct way to proceed is as follows: In the gathering of statistics (as described in (3.2) or (3.5)), both $rMAXH_i$, $i = 1, 2$, are updated; use the $rMAXH_i$ so obtained to define $\hat{rh}^t_{iM}$ by (3.1); finally, in analogy with (2.3), set

(3.6) $\hat{rh}^t_{(1,2)M}(n) = \max\{\hat{rh}^t_{1M}(n), \hat{rh}^t_{2M}(n)\}$, for all $n \in G$.

In analogy with (3.4), one has, when both $rMAXH_i$ have grown sufficiently, that

$$\hat{rh}^t_{(1,2)M}(n) \geq h^t_{(1,2)M}(n) \quad \text{for all } n,t \in G.$$

The technique extends to more than two heuristics in an obvious way.

If one of the h_i, say h_1, is already admissible, one could use a variant of (3.6) obtained by replacing $\hat{rh}^t_{1M}$ with h^t_1. As pointed out in Section 2, this substitution is equivalent to using h^t_{1M} when $MAXH_1$ has its values on the 'diagonal'. Convergence of A^* to an admissible algorithm is faster in this case.[5]

3.4 A probabilistic Learning Theorem.

Will the learning described above eventually cause only optimal solutions to be found? The theorem below shows that, in a probabilistic sense, the answer is 'yes' • Proof is in the appendix.

Theorem 3.1 (Probabilistic Learning Theorem). Assume G is finite and that P_1, P_2,... are randomly, and independently, generated problems from *GxG.* Assume *A^(rh_M)* is using any one of the learning techniques described above. With probability *1,* there exists *i* such that after P_1,...,P,. are solved we have: $rh^t = h^tM$ for all t E G.

Hence, *A*(rh_M)* is admissible from some point on.

4, Experiments with Quadratic Learning

The effectiveness of learning may vary with the problem domain. To get some idea of what to expect, we conducted experiments in the 8-puzzle using the quadratic learning method. We used the Manhattan distance, h_2, along with two non-admissible heuristics: h_3, the enhanced Manhattan distance, and h14. Briefly, *h.4* adds to *h2* weighted row-column and diagonal terms: the former counts the number of interchanged tile pairs which are in the proper row or column; the latter counts the number of tiles that are diagonally displaced and are blocked by in-place tiles. While *h3* almost always overestimates and seldom finds optimal solutions, *h4* normally under-estimates and usually, but not always, finds optimal solutions.[6]

Our experiments were with *rh3M, rh4M,* rh(2,3) and *rh(2,4)M-* We had two sample sets: the first consisted *of 1998* randomly generated problems, and the second consisted of the initial *605* problems of the first. We sometimes reordered them to study how this affected experimental results.

4.1 Speed of Learning

We cannot know when the true *MAXH* has been learned because we do not know its value. However we can get an assesment of learning speed in the following way: Generate a large sample of random problems and record *rMAXH* when *A*(rh_M)* is through solving the set of problems; now compare this *rMAXH* with the *rMAXH* which was learned at various 'snapshot points' during the problem solving session. This will show us how quickly the learning procedure acquired its final version of *rMAXH.*

In the 8-puzzle, learning occurs quickly. For example, Figure 4.1 shows snapshots of how much *A*(rh_3M)* learns while solving a set of *1998* randomly generated problems. After one problem *rMAXH* has been learned for distances *1* to *8* and after *8* problems through distance *17.* Furthermore, at this point its knowledge of *rMAXH* for the remaining distances is within 6% of its final values. It acquired no more knowledge after executing 1332 problems.

The pattern of Figure 4.1 was consistently observed: When the problems are randomly sorted, the vast amount of learning occurs within *8* problems. One could speed the learning process up by putting hard problems *(ie.,* large start-goal distances) at the beginning and slow it down by putting easy ones first.[7]

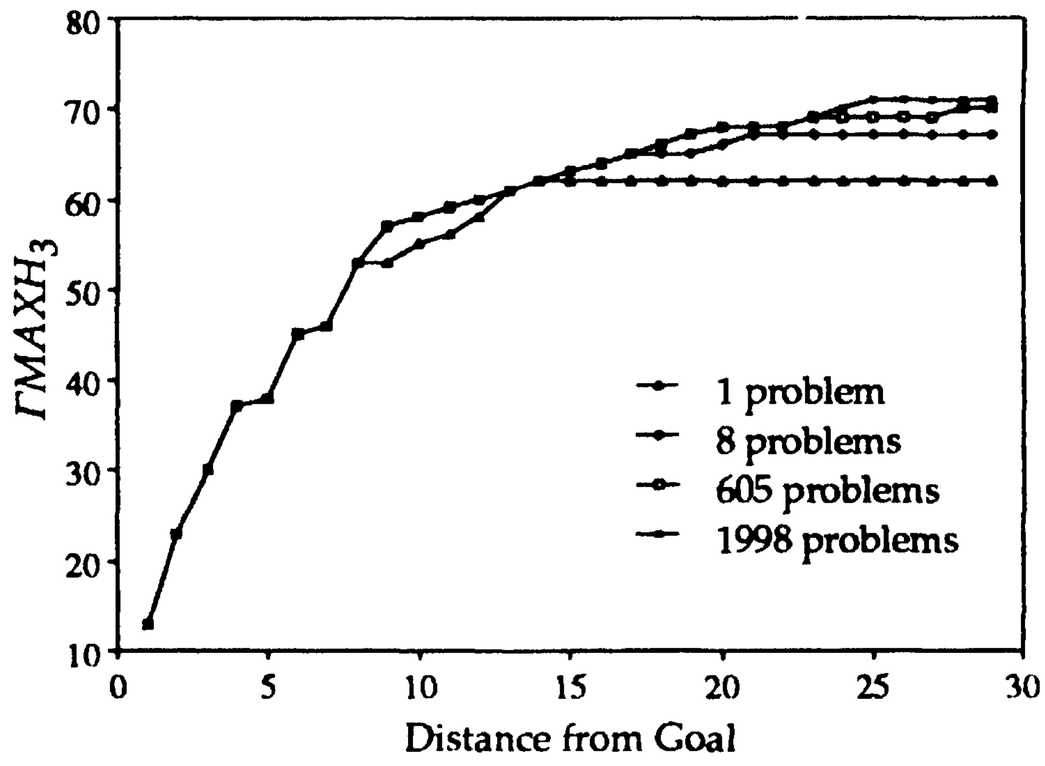

Figure 4.1. Extent to which *OAAXH^is* learned after *1, 8, 605,* and *1998* problems. Most information is learned after *8* problems. Quadratic learning is being performed during searches by *A ^(TH^M).*

4.2 Solution Quality while Learning: Observation and Theory

We observed high solution quality while learning on randomly sorted problems sets: In our experiments *A** returned an optimal solution in all but the first problem. This could not be guaranteed and surprised us. When the problems are arranged with the easy ones first, we observed that *A** returned several non-optimal solutions.

In the proof of the learning theorem (see appendix) it is brought out that we may expect *MAXH(x)* to be learned

[5] If $MAXH_1$ is not diagonal, then the heuristic converged to, while admissible, is not the same as *h(1,2)M* and may be weaker. See discussion in Section 2.

[6] h_4 is a variation of a heuristic discovered semi-automatically by Politowski [19861.

[7] In the experiments reported here the problems are randomly sorted and the first has start-goal distance of *19.* This is considered a relatively easy problem because in the 8-puzzle the mean optimal distance between states is *22.*

first for low values of x and then for higher values. This is the pattern shown in Figure 4.1 by the four successive snapshots of the knowledge acquired. Of course we don't know with certainty whether or not the true $MAXH(x)$ is learned. When it is learned for all $x \leq x_0$, say, then, according to the appendix, $A^*(rh_M)$ returns optimal solutions for all problems whose start-goal distance is less or equal to x_0. The observed behavior of $A^*(rh_M)$ suggests that the learned $MAXH$ may, in fact, be correct.

4.3 Early learning to build a composite heuristic

Suppose that, during a problem solving session with $A^*(rh_M)$, we stop the learning before all problems are solved. The heuristic which has evolved up to this turn-off point is said to have been acquired by *early learning*. Experiments were conducted to see how effectively early learning could be used to build a composite heuristic from two others. Our general finding is that, by early learning, the composite heuristic acquired is often one which produces high quality solutions at low cost. Cost is measured as number of nodes expanded per problem. We give two examples below.

Learning was allowed on only the first problem of the *605* sample set using *A*(rh(2t4)M)- A* was then run on* the whole sample set using the heuristic learned in the first problem. See Table 4.1. Solution quality was within 6% of optimal and half as many nodes were expanded as were expanded by *A*()i2).* The experiment shows that using early learning to add the insight of the non-admissible heuristic *h^* to that of *hj* creates a heuristic that reduces the cost of A* while making only limited compromises on solution quality.[8] On harder problems the cost reduction is higher (72%). Similar, but less spectacular results for *A (rh(2j)M)* are also shown in Table 4.1.

In the second experiment *A*(rh(2f4)M)* was allowed to learn on the first eight problems. Then, considering the entire sample set of 605 problems, we compared the performance of *A** using the heuristic learned in eight problems with the performances of $A^*(h^\wedge)$ and $A^*(h_2)$. The result is shown in Table 4.2. *A*(rh(24)M)* produced perfect solution quality and expanded substantially fewer nodes then did the other algorithms: *15%* fewer than *A*(h2>* and *48%* fewer than $A^*(h_4)$, with even better performance on the hard problems.

Once the learning process is turned off, the use of *m(2,4)M* requires only a little more time per node then the sum of that required to evaluate both h_2 and h_4; the extra time is for lookups into two tables of size bounded by the diameter of the graph, and a miximizing process. The resulting heuristic causes high quality solutions and relatively low node expansion count. In a general setting, the use of a composite heuristic like *rh^^M* is justified if the computational time for evaluating both h_2 and h_4 is sufficiently low.

		average performance					
No. of problems		$A^*(rh_{(2,3)M})$			$A^*(rh_{(2,4)M})$		
		SQ	*X*	$\%X_2$	*SQ*	*X*	$\%X_2$
easy (1-20)	187	1.02	170	96	1.03	152	86
med (21-25)	346	1.05	645	79	1.07	496	61
hard (26-29)	72	1.04	1236	51	1.07	679	28
All (1-29)	605	1.04	569	70	1.06	411	51

Table 4.1 Performance comparison of $A^*(rh_{(2,3)M})$ and $A^*(rh_{(2,4)M})$ with $A^*(h_2)$. Early learning has occurred on one randomly chosen problem. Its start goal distance was *19*. X denotes number of nodes expanded. *SQ* is solution quality (defined in Section 2) and is *1.00* for $A^*(h_2)$.

		average performance			
No. of problems		$A^*(r_8h_{(2,4)M})$			
		SQ	*X*	$\%X_2$	X_4
easy (1-20)	187	1.00	154	87	56
med (21-25)	346	1.00	704	86	54
hard (26-29)	72	1.00	2012	83	48
All (1-29)	605	(*)1.00	690	85	52

Table 4.2 Performance comparison of $A^*(r_8h_{(2,4)M})$ with $A^*(h_2)$ and $A^*(h_4)$. Note (*): All solutions found were optimal but there is no guarantee that this will always happen. Early learning has occurred on *8* problems.

		average performance					
No. of problems		$A^*(h_2)$		$A^*(h_3)$		$A^*(h_4)$	
		SQ	*X2*	*SQ*	*X3*	*SQ*	*X4*
easy (1-20)	187	1.00	178	1.22	104	1.00	277
med (21-25)	346	1.00	818	1.24	166	1.00	1312
hard (26-29)	72	1.00	2427	1.20	208	1.00	4187
All (1-29)	605	1.00	812	1.22	152	(*)1.00	1334

Table 4.3. Performances of $A^*(h_2)$, $A^*(h_3)$ and $A^*(h_4)$ are given for comparison with previous data. h_4 causes strong solution quality but poor time complexity. The opposite is true of h3. The text assumes that the solution quality of $A^*(h_3)$ is unacceptable. Note (*): All but *10* solutions were optimal; *1.00* results from averaging to two decimal places.

5. Conclusion

It is possible to find optimal solutions with a non-admissible heuristic *h* by letting *A** use *g* + *h^* as an evaluator, where h_M is an admissible heuristic associated with *h;)%M* contains much of the 'insight' that *h* has. There is a difficulty in calculating *h^* because it is based on an upper bound function for *h* which is hard to access.

We have described a technique for solving a set of problems using $A^*(rh_M)$, where rh_M is a dynamically changing approximation of *h^*. As problems are solved, statistics are gathered enabling *rh^* to evolve from *h* to *h^*. In the process, *A** returns solutions of increasingly better quality. We proved that, in a probabilistic sense, *rh^* converges to *h^*, causing *A*(rh^)* to be admissible after a finite number of problems have been solved.

The above ideas extend to the case in which, instead of

* *A** using /14 alone expands many more nodes than does $A^*(h_2)$. See Table 4.3.

[9] As /14 is usually optimistic it is tempting to combine h_2 and h_4 by forming the heuristic $h_24 = max(h_2, h.4)$. One could then compare the performance of *A** using h_24 with that of $A^*(rhQ4)_M$. But it turns out that one always has $h_4 > h_2$ so *\%24 = h.4.* Thus there is no need to consider $A^*(h_24)$.

a single heuristic h, one starts with a finite set of heuristics, say $h_1,...,h_p$. The admissible heuristic, h_M, to which we now have convergence, is a composite of $h_1...,h_p$ (and contains their 'collective insight').

To gain some empirical understanding of this type of learning we performed experiments in the 8-puzzle domain using a variation of our technique called 'quadratic learning'. Learned information was acquired surprisingly quickly. The preponderance of information acquired in many hundreds of random problems was actually learned after solving only eight of them. Although we always started with an overestimating heuristic, the system learned so fast that all but the first problem were solved optimally.

In another experiment, we used 'early learning' to combine an admissible heuristic with a non-admissible one. The goal was to create a heuristic which caused A^* to expand significantly fewer nodes than when it used the admissible heuristic alone; but nevertheless we wanted A^* to yield high quality satisficing solutions. We were able to achieve this. In one case, for example, we obtained a reduction in node count of nearly half while only losing 6% in solution quality.

Our results show that it can be fruitful, while solving problems, to learn the statistical properties of the heuristic guiding the search. Knowledge of these properties may then be used to alter the heuristic itself, bestowing it with traits suitable to the application need. Moreover altering the heuristic can be done while the search is ongoing.

Appendix. Proof of Theorem 3.1

We give the argument for constant learning. This implies the linear and quadratic case because these learn at least as much in each problem as constant learning. For notational simplicity we do not consider the combined heuristic case of Section 3.3. The argument in that case is similar.

Let $[a,b]$, $[a,b)$ denote $\{x: x\epsilon R^+, a \le x \le b\}$ and $\{x: x\epsilon R^+, a \le x < b\}$ respectively. The theorem follows from the two lemmas below. We omit the proof of Lemma A.1, as it is clear.

Lemma A.1. Suppose $u > 0$ and that at the end of some initial finite sequence of problems we have $rMAXH(x) = MAXH(x)$ for all $x \epsilon [0, u]$. Then in any future problem (s,t) we have

$$rh^t_M(n) = h^t_M(n) \qquad \text{whenever } n \epsilon G \text{ and } k(n,t) \le u.$$

Lemma A.2. Let $w_1 < w_2 < ... < w_p$ be the discontinuity points of $MAXH$. Let $w_0=0$ and let $w_{p+1} \ge w_p$ be the length of the largest optimal path in G. Then given any $j\epsilon\{0,1,...,p+1\}$ there exists, with probability 1, some i such that after $P_1,...,P_i$ are solved by $A^*(rh_M)$ we have for all future problems that

(A.1) $rMAXH(x) = MAXH(x)$ whenever $x\epsilon [0,w_j]$

Proof: Figure A.1 illustrates the situation. We use induction on j. If $j = 0$, then (A.1) is clear. Assume (A.1) is true for $j = k\text{-}1$. We must show it is true for $j = k$. Suppose the truth of (A.1) in the $j = k\text{-}1$ case occurs after $P_1,...,P_{i_1}$ have been solved.

If $u > w_{k-1}$ and $MAXH(u) = MAXH(w_{k-1})$ then (A.1) holds with w_j replaced by u; this is because both $rMAXH$ and $MAXH$ are non-decreasing, they agree at w_{k-1}, and $rMAXH \le MAXH$. If $k\text{-}1 = p$ and w_{p+1} is not a discontinuity point (it is not in Figure A.1), then we are done. Assume that w_k is a discontinuity point. Then, after solving $P_1,...,P_{i_1}$, we have that for all future problems

(1) $rMAXH(x) = MAXH(x)$ *for all* $x\epsilon [0,w_k)$

We must show that, after solving more problems, the truth of (1) extends to w_k.

Let $(s,t)\epsilon GxG$ satisfy $k(s,t) = w_k$ and be such that $H(s,t)$ is the largest number in $\{H(m,n)$: $m, n\epsilon G, k(m,n) = w_k\}$. As the problems P_j are chosen randomly and independently from GxG, the probability of (s,t) never being chosen after $P_1,...,P_{i_1}$ is zero. Thus, with probability 1, there is some $i > i_1$ such that $P_i = (s,t)$. We shall show that after solving P_i we have

(2) $rMAXH(x) = MAXH(x)$ for all $x\epsilon [0,w_k]$.

This will complete the induction and establish the lemma.

Let $s= n_o,n_1,...,n_q = t$ be an optimal path for P_i. In lemma A.1 set $u = k(n_1,t)$ to conclude that rh^t_M is optimistic on all the nodes $n_1,...,n_q$ during the solution of P_i, because of (1). After the first iteration of $A^*(rh^t_M)$, when n_o is removed from Open, there will be a least r, $1 \le r \le q$, such that $n_r\epsilon$ Open. For this node, $g(n_r)+rh^t(n_r) \le w_k$, since $n_o,...,n_q$ is an optimal path. It follows that $A^*(rh^t_M)$, selects from Open, upon halting, an optimal goal for P_i; that is, one whose g-value is w_k. At this point, the constant learning procedure correctly updates $rMAXH(w_k)$ so that (2) holds. Δ

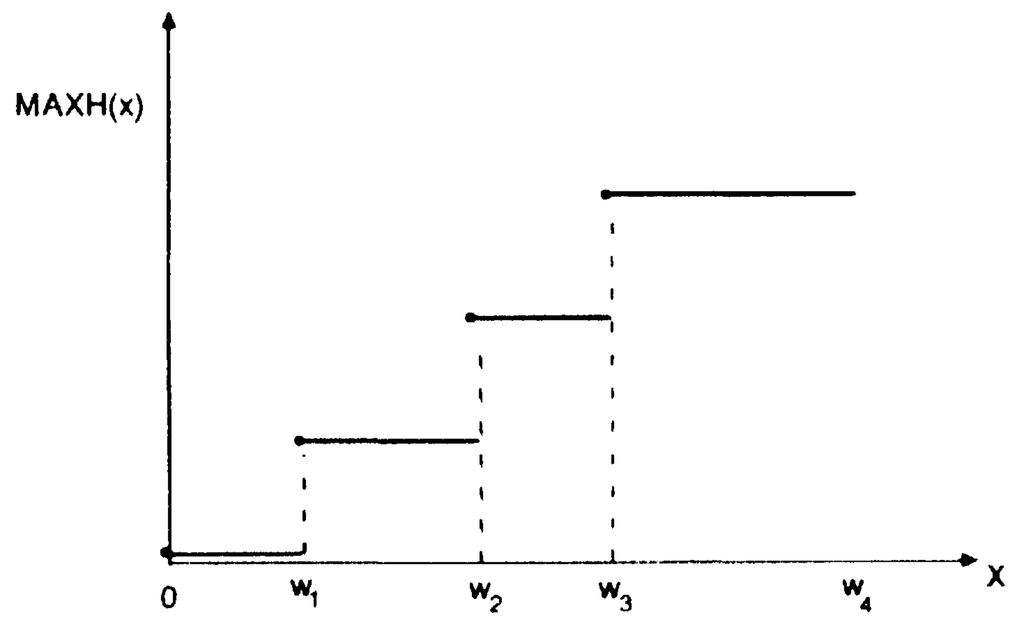

Figure A.1. Possible graph of $MAXH$: $p = 3$, $w_4 > w_3$. In this case w_4 is not a discontinuity point but, in general, it might be.

References

[Davis *el al.*, 1989] Davis H.W., Bramanti-Gregor A., and Chen X., Towards finding optimal solutions with non-admissible heuristics: a new technique, *Proceedings of the 11th International joint Conference on Artificial Intelligence,* 303-308, 1989.

[Nilsson, 1980] Nilsson N.J., *Principle of Artificial Intelligence,* Tioga Publishing Co., Palo Alto, Ca., 1980.

[Politowski, 1986] Politowski C, *On the Construction of Heuristic Functions,* Ph.D. Thesis, University of California, Santa Barbara, 1986.

IRatner *el al.,* 1986] Ratner D. and Warmuth M., Finding a shortest solution for the *NxN* extension of the 15-puzzle is intractable, *Proceedings of the 5th National Conference on AI* (1986) 168-172.