Sādhanā, Vol. 21, Part 3, June 1996, pp. 291–315. © Printed in India.

Learning for efficient search

SUDESHNA SARKAR†, SUJOY GHOSE and P P CHAKRABARTI

Department of Computer Science & Engineering, Indian Institute of Technology, Kharagpur 721 302, India
†Present address: Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati 781 001, India
email: sudeshna@iitg.ernet.in; {sujoy,ppchak}@cse.iitkgp.ernet.in

Abstract. Learning for problem solving involves acquisition and storage of relevant knowledge from past problem solving instances in a domain in such a form that the information can be used to effectively solve subsequent problems in the same domain. Our interest is in the role of learning in problem solving systems that solve problems optimally. Such problems can be solved by an informed search algorithm like A*. Learning a stronger heuristic function leads to more effective problem solving. A set of arbitrary features of the domain induce a clustering of the state space. The heuristic information associated with each cluster may be learned. We discuss the use of a new form of information in the form of *h*set* (the set of optimum cost values of all nodes of the cluster) and present an algorithm for using the information that is more effective than A*. A possibilistic (fuzzy set theoretic) extension of this algorithm is also presented. This version can handle incomplete information and is expected to find solutions faster in the average case with controlled relaxation in the optimality guarantee. We also discuss how to make the best use of the features, when the system has memory restrictions that limit the number of classes that can be stored.

Keywords. Learning; best first search; features; heuristic estimate; clustering.

1. Introduction

1.1 *Search and A**

A search problem consists of finding a sequence of operators to transform a given initial state to a state matching the given goal description such that the sum of the costs of the operators along the path is minimum (Nilsson 1980). For uninformed systematic search strategies like depth first search and breadth first search and their variants, the number of node expansions is an exponential function of the optimum solution length (Pearl 1984).

In order to reduce the number of node expansions, informed search algorithms use some domain dependent information of a node, known as a heuristic estimate of the node, usually in the form of a function of the node. Under certain assumptions on the value of the heuristic estimate, these estimates can be suitably used by search algorithms for reduced search complexity. A* is an informed search algorithm, that uses an estimate (underestimate) of the actual solution cost of the node as the heuristic information. A* guarantees that the goal found will be the one with an optimum solution cost when the heuristic information is an underestimate. If A* has access to perfect information, the number of nodes expanded will only be linear in the solution cost.

1.2 *Role of learning*

Most problem solving systems are expected to solve problems repeatedly from the same domain and often the same population of problems over their lifetime. In the course of problem solving an intelligent system can be expected to acquire additional experience or knowledge that will help it to solve subsequent problems of the domain more effectively. The time taken by best first search algorithms depends on the accuracy of the heuristic estimate. Therefore learning a more accurate cost function associated with a state will help to speed up the search process.

For learning to be effective, we make the following assumption: *problems come from the same arbitrary but fixed distribution in the learning phase as in the problem solving phase*. This is the framework of distribution free learning, and we will claim that if the underlying distribution is the same in both cases, with high probability, the information acquired by the system in the learning phase will be sufficient to solve most of the problems in the problem solving phase effectively.

1.3 *Background*

In traditional heuristic search, a heuristic function is used as a direct estimate of the cost of a node. This restricts the types of features that can be used as information. Efficient performance of an admissible heuristic search algorithm depends on finding a good easy way to compute underestimating feature. Such features are often hard to find.

A major problem with the use of heuristic functions is that the information content of an individual feature may not be very high. Often we come across features that capture some aspects of the problem domain only. Several features taken together can characterize a problem domain much better than any of the features taken individually by providing more discriminatory information. Use of multiple features has been explored by several researchers. Given a set of features, best first search algorithms traditionally compute the values of the corresponding heuristic functions and combine the values to yield the heuristic estimate of a node. The appropriate combination of the different feature values that will yield a more informed estimate of a node, while leaving the search procedure admissible, is an interesting problem. The standard form of combination is a linear polynomial (Samuel 1963; Christensen & Korf 1986; Lee & Mahajan 1988; Bramanti-Gregor & Davis 1993). Learning in this context is used to determine optimal coefficients of such a polynomial. This method may not yield good results when there is no effective linear mapping from feature

vector values to the actual cost of nodes. A linear discriminator of the features is very restrictive and some form of nonlinearity has been proposed (Samuel 1967). Bramanti-Gregor & Davis (1993) present a method of combining features based on linear regression. They propose a linear combination of certain *ad-hoc* nonlinear forms of features.

Politowski (1986) has used a model where the feature values are used to partition the state space into a number of equivalence classes. He advocates that the information corresponding to a class be explicitly represented as a mapping from the class to the information. This mapping can be learned during the training phase. We adopt this model as our basic model.

1.4 *Model*

The state space is first divided into classes by the features. Information associated with the classes is learned during the learning phase, and the acquired information is used to solve problems more effectively in the problem solving phase.

Classification
Following Politowski (1986), we look upon the features as discriminators that determine the cost of a node. A feature, h, of a problem domain is a function which can assume a number of distinct values when applied to different states of the problem. When multiple features of a problem domain are available, a feature set can be used which will provide a vector of values for every state. Suppose that we have m features of the problem domain $\mathbf{k} = \langle h_1, h_2, \ldots, h_m \rangle$.

The feature vector values induce a clustering of the state space. Each k-element vector identifies a class containing some number of nodes that are heuristically indistinguishable from each other. Consider the set $S_{\mathbf{k}}$ of all nodes which have the identical value, $\mathbf{k}$ as their feature vector value.

$$S_{\mathbf{k}} = \{n : hvec(n) = \mathbf{k}\}.$$

We advocate that information be associated and stored with each such set $S_{\mathbf{k}}$, to be used during a best first search algorithm.

Information associated with a class
Corresponding to any specific value of the feature vector $\mathbf{k}$ at a node, we may store the minimum of the h^* values of all elements of the class. We may also store the set of h^* values of the elements of a class, represented by $h^*set(\mathbf{k})$.

$$h^*set(\mathbf{k}) = \{v : h^*(n) = v, n \in S_{\mathbf{k}}\}.$$

Algorithm
The h^*set corresponding to the value of the feature vector at the node can provide more information to an informed best first search algorithm than what can be provided by a single underestimate value. We present algorithms CS* , FS* and their variants that use h^*set and its distribution as information and performs better than A* using the corresponding single-valued estimate.

1.5 *Work done*

- Given any set of features, the minimum element of h^*set can be used as a heuristic underestimate at a node that can be used by the algorithm A*. In this framework, we show how the use of **multiple features** drastically reduces the search time, while solving problems optimally.
- Given a set of features of the problem domain, we show that the h^*set corresponding to each feature vector value can be learned in the limit by solving a sufficient number of example problems in the training phase. We have devised **best first search algorithms** that make effective use of the information of the entire **h*set** to further reduce the number of node expansions.
- However these algorithms assume complete information about the h^*set corresponding to every feature vector value. A bounded risk model has been devised that will always find some solution, and is guaranteed to find an optimum solution with a specified confidence level. The algorithm uses the possibility distribution of the individual elements of the h^*set .

While a lot of stress has been laid in previous work on the importance of accuracy and precision as the necessary quality of a good heuristic estimator, making use of the discriminatory power of the heuristic feature set has been largely overlooked. We lay special emphasis on devising more effective best first search algorithms by making use of the discriminatory power of a set of features that may help to make a distinction between *on-track* and *off-track* nodes.

This paper is organized as follows:

Section 2 provides the basic notations and definitions. Section 3 discusses the learning phase. Section 4 presents admissible search algorithms for effective use of the information learned. Section 5 provide the corresponding algorithms for the fuzzy set model. Section 6 concludes the paper.

2. Features, feature vectors and h^*set

Let $\mathcal{S}$ be the set of states of the *problem space*. The problem space has a set of operators that change the state of the problem. For example, in 8-puzzle, the states are the different permutations of tiles, and the operators are single step moves of the blank position. Corresponding to each application of an operation in moving from state s_i to state s_j, a cost, $c(s_i, s_j)$ is associated.

A search problem instance is an element of $\mathcal{P}$, where $\mathcal{P} = \mathcal{S} \times \mathcal{S}$. In other words, a search problem is a 2-tuple (s, γ). s is called the start node, and γ is the goal node. A problem can also be of the form (s, Γ) where Γ is a set of goal states. In optimum search, the problem solving task is to find a sequence of operators that map the initial state to one of the goal states such that the sum of the costs of the operators along the path is minimum.

First we provide certain preliminary definitions:

s : A node designated a start node for a specified problem.

γ : A goal node for a specific problem.

Γ : A set of goal nodes for a specific problem.

$(\mathbf{n_i}, \mathbf{n_j})$: Arc connecting the node n_i and the node n_j.

$\mathbf{c(n_i, n_j)}$: Cost of the arc connecting the node n_i and the node n_j.

$\mathbf{P = (n_0, n_1, \ldots, n_k)}$: Path in a graph consisting of a sequence of arcs connecting n_0 through n_k.

$\mathbf{P \mid n}$: If $P = (n_0, n_1, \ldots, n_k)$, $P \mid n = (n_0, n_1, \ldots, n_k, n)$.

$\mathbf{P_{n_1-n_2}}$: A path from n_1 to n_2. $P_{n_1-n_2} = (n_1, \ldots, n_2)$.

$\mathbf{PP_{s-n}}$: Pointer Path in a graph obtained by tracing the pointers back from n to s in the current search tree. It is the cheapest known path from s to n at any given point during the search.

$\mathbf{C_p(P)}$: Cost of path P. If $P = (n_0, n_1, \ldots, n_k)$, $C_p(P) = \sum_{i=0}^{i=k-1} c(n_i, n_{i+1})$.

$\mathbf{C_p^*(n_1, n_2)}$: Cost of minimal cost path from n_1 to n_2.

$\mathbf{g^*(n)}$: Cost of cheapest path from the implicit start state s to the node n. $g^*(n) = C_p^*(s, n)$.

$\mathbf{g(n)}$: Cost of cheapest known path from the start state s to the node n at any given point during the search.

$$g(n) = min_P(C_p(P)) \ \forall \ P = (s, \ldots, n).$$

In other words, $g(n) = c_p(PP_{s-n})$.

$\mathbf{h^*(n)}$: Cost of cheapest path from n to the goal state γ of the given problem. $h^*(n) = c_P^*(n, \gamma)$ where γ is the implicit goal. It may be noted that the goal may vary from problem to problem. However, for notational convenience, γ has been dropped.

$\mathbf{f(n, P)}$: Estimate of cost of cheapest solution path which is an extension of path PP_{s-n}.

$\mathbf{h}$: A feature of the problem domain.

$\mathbf{k} = \langle \mathbf{h_1}, \mathbf{h_2}, \ldots, \mathbf{h_k} \rangle$ The set of features used (feature vector). $h_1, h_2, \ldots, h_k$ are features.

$\mathbf{v} = \langle \mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_k} \rangle$ Instantiated value of the feature vector, where v_i is the value of h_i for the given state.

$\mathbf{hvec(n)}$: The feature vector value corresponding to node n. $hvec(n) = \langle v_1, v_2, \ldots, v_k \rangle$ if $v_i = h_i(n)$.

2.1 *What are features?*

A heuristic feature is traditionally an estimate of the cost of an optimum cost path from the current state to the goal state of the current problem instance. We consider a feature to be any function of the goal state γ, the start state s, and the partial search tree generated so far. However, for convenience, we will use $h(n)$ to denote the value of a feature h at node n.

2.2 *Role of features in clustering*

We will show how a set of features can be effectively used to strengthen the heuristic estimate of a node. We look upon features of the problem domain as a means of dividing the problem state space into a number of classes. Increase in the number of features leads to further refinement of the classes, each original class being subdivided into a number of classes based on the values of the new feature set. Instead of associating information with a state, information can be associated explicitly with every class. We shall see that with the types of information that we consider, the more refined the classes, the greater the accuracy of the information.

Example 1. Consider the 8-puzzle problem. The number of distinct states is about 181440 (9!/2). The misplaced tiles heuristics takes on 9 different values $0, \ldots, 8$, and divides the state space into 9 different classes. Similarly the Manhattan distance estimate divides the state space into about 25 different classes. Taken together, they divide the state space into about 78 different classes. The information associated with these subclasses induced by a set of features is stronger than the information that can be associated with classes induced by each of the component features. □

Note that the above scheme gives rise to a finite number of classes only for features which take on a finite number of distinct values. To be able to handle features that take on continuous values, or a large number of values, we will have to divide the feature values into intervals. The feature vector in this case will be a vector of intervals, rather than a vector of values.

2.3 *minh and h*set*

After classification, some information must be associated with the classes, which can be used as the heuristic information for the nodes belonging to the class. Search algorithms use estimates of the optimum cost as the heuristic information. A class may contain nodes with several different optimum cost values. The simplest form of information is the minimum possible cost value corresponding to a class, denoted by *minh*. We introduce the concept of *h*set* associated with a class.

DEFINITION 1
The *minh* of a given feature vector value $\mathbf{k}$ is the minimum of the h^* values of the nodes comprising the class identified by the feature vector value $\mathbf{k}$.

$$minh(\mathbf{k}) = min_n\{h^*(n)\} \quad \forall n \;\; s.t. \;\; hvec(n) = \mathbf{k}.$$

DEFINITION 2
The h^*set of a given feature vector value $\mathbf{k}$ is the set of feasible optimum cost values i.e., $h^*(n)$ for all nodes with feature vector value $\mathbf{k}$ and is denoted by $h^*set(\mathbf{k})$.

There will exist many nodes $n_1, n_2, n_3, \ldots$ which have the same feature vector $\mathbf{k}$ i.e.,

$$hvec(n_1) = hvec(n_2) = hvec(n_3) = \cdots = \mathbf{k}.$$

Table 1. Examples of h^*set in 8-puzzle.

Feature set	Feature vector value	*minh*	h^*set
$\langle h_1 \rangle$	$\langle 10 \rangle$	10	10 12 14 16 18 20 22
$\langle h_1, h_2 \rangle$	$\langle 10, 8 \rangle$	10	10 12 14 16 18
$\langle h_1, h_2, h_3 \rangle$	$\langle 10, 8, 8 \rangle$	12	12 14
	$\langle 10, 8, 9 \rangle$	10	10 12 16
	$\langle 10, 8, 11 \rangle$	10	12 14
	$\langle 10, 8, 12 \rangle$	16	16
	$\langle 10, 8, 13 \rangle$	14	14 16
	$\langle 10, 8, 15 \rangle$	18	18

These different nodes may have different values of h^*. The h^*set corresponding to the feature vector is the set of the h^* values for all nodes with that feature vector i.e.,

$$h^*set(\mathbf{k}) = \bigcup_n h^*(n) \quad \forall n \ \ s.t. \ \ hvec(n) = \mathbf{k}.$$

Corresponding to every element x of $h^*set(\mathbf{k})$, there will exist some node n with feature vector value $\mathbf{k}$ and optimum solution cost $= x$.
Note that $minh(\mathbf{k}) = min\{h^*set(\mathbf{k})\}$.

Given a node n, we will, for convenience, refer to the *h^*set at the node, n* to mean the h^*set corresponding to the feature vector value at the node $(h^*set(hvec(n)))$.

Example 2. In table 1 we show some typical examples of h^*set values in the domain of 8-puzzle. The example illustrates how the addition of more features induces a finer clustering of the state space, and stronger information may be associated with these subclasses. The heuristic features used are

- *Manhattan distance*[1] represented by h_1,
- *Misplaced tiles*[2] represented by h_2,
- *Sequence count*[3] represented by h_3, □

3. Learning phase

This phase must precede problem solving. In this phase, the mapping from feature values at states to the corresponding estimates are learnt. In this section, we discuss the number of problems that must be solved in this training phase for effective problem solving using

[1]Manhattan distance: It is the sum of distances that each tile is from home and is denoted by manhattan.
[2]Misplaced tiles: The number of tiles not in their correct position as in the goal state.
[3]Sequence count: It is obtained by checking around noncentral squares in turn, allotting 2 for every tile not followed by its proper successor, and allotting 0 for every other tile; a piece in the centre scores 1. We denote this by sequence count.

this information. In Bramanti-Gregor & Davis (1993), the mapping function from features to estimate is assumed to be linear. If a close linear fit does exist to the actual function, the equation of the line can be obtained by using a few examples for training. We need to learn the function without assuming that the function is to be approximated to any particular form.

Let $\langle h_1, h_2, \ldots, h_k \rangle$ be a given set of features of the domain. The system solves representative problems in the domain, using some guaranteed admissible search algorithm; and learns the full range of feasible h^* values (optimum cost from a node to the goal node), $h^*set(\mathbf{k})$, corresponding to all nodes having the feature vector **k**.

For a particular start state s, a necessary condition that our algorithm will terminate with an optimum solution is that there exists an optimum cost path from s to γ, such that corresponding to every state in the path, some state was encountered in the learning phase with the same feature vector value and the same value of h^*.

We define $S(\mathbf{h}_i, h_i^*)$ to be the equivalence class of states containing all nodes n s.t.

$$\forall n \in S(\mathbf{h}_i, h_i^*), \quad hvec(n) = \mathbf{h}_i \ \ \& \ \ h^*(n) = h_i^*.$$

Let $P_{s-\gamma}^*$ be an optimum path from s to γ, given by

$$P = \langle n_0 = s, n_1, n_2, \ldots, n_k = \gamma \rangle .$$

The necessary condition that this path will be found in the problem solving phase is: For all i, $0 \leq i \leq k$, at least one member of $S(hvec(n_i), h^*(n_i))$ was encountered in the problem solving phase.

Suppose we are working with a feature set $< h_1, h_2, \ldots, h_k >$, which induces M clusters in the state space. These clusters are represented as $C_0, C_1, \ldots, C_M$. We run a number of example problems and obtain T points which are used for learning. We assume a model where the examples come from any arbitrary distribution, from a given population of problems. The distribution from which problems come from is assumed to be the same in the problem solving phase as in the learning phase. To be able to learn the class C_i it is required that the set of T sample points contain some examples of class C_i. Suppose the probability distribution is not uniform, but states occur according to some arbitrary but fixed distribution D^+. In this case perfect learning of the h^*set is guaranteed only in the limit. But suppose we relax the guarantee of completeness of learning — and only require that with confidence $> (1 - \delta)$ we want to get the optimum solution. The class C_i will be correctly learnt if

- for A*(*minh*), an example is found where a node belonging to class C_i has its h^* value as $minh(hvec(C_i))$.
- for CS* : for each x in $h^*set(C_i)$, an example is found where a node belonging to class C_i has a h^* value of x.

The class C_i will be approxiamtely correctly learnt if

- for A*(*minh*), an example is found where a node belonging to the class C_i maps to the h^* value of ε cut of the distribution corresponding to $hvec(C_i)$.
- for CS* : for each x in $h^*set(C_i)$ where the probability of the point x is $> \varepsilon$, an example is found where a node belonging to class C_i has the h^* value of x.

Let the probability of occurrence of a class be $> \varepsilon$. Let N be the maximum length of an optimum solution path; and let X be the number of trials necessary for the $\langle\delta\rangle$ guarantee. Let the number of sample points be T. Then the probability that at least one of the classes containing some state in the optimum path was not encountered in the X trial phases is $N(1-\epsilon)^X$. The δ confidence guarantee means that

$$\begin{aligned} & N(1-\varepsilon)^X < \delta \\ \Longrightarrow \quad & Ne^{-\varepsilon X} < \delta \\ \textit{i.e.,} \quad & e^{\varepsilon X} > N/\delta \\ \textit{or,} \quad & X > (N/\varepsilon)\log(N/\delta). \end{aligned}$$

Therefore if we want to find optimum solutions with a $\langle\delta\rangle$ guarantee, it suffices to have collected data for h^*set from $O(\frac{1}{\varepsilon}\log\frac{N}{\delta})$ random examples from the same distribution as the example problems.

Let us look at classes with probability $< \varepsilon$.

Maximum probability that a class falls in one of these classes $= T\varepsilon$.

The probability of that at least one of the states on the solution path has probability $< \varepsilon$ is given by $= TN\varepsilon$.

We want this probability to be less than δ.

This is true if $\varepsilon < \delta/TN$, i.e., $1/\varepsilon > TN/\delta$.

Therefore, if we look at

$$(TN/\delta)\log(N/\delta)$$

examples, we can ensure that the probability of failure will be $< \delta$.

Once we are able to learn the information corresponding to every feature vector value, we are ready to use it to solve future problems more effectively.

4. Perfect information model with crisp h^*set

In this section, we discuss algorithms making use of the information learnt to perform admissible search. In § 4.1, we discuss using of *minh* as the heuristic evaluation function in A*. Results are provided for the domain of 8-puzzle which serves as a benchmark for comparison of the performance of the other algorithms.

4.1 *Using* minh *as information*

The simplest type of information that we can associate with every class is an estimate of the cost value of the nodes belonging to the class. To be able to run algorithm A*, the best underestimate that can be associated with any cluster, C_1, with feature vector value $\mathbf{k}_1$, is the minimum of the h^* values of the nodes comprising the cluster. This is the scheme used by Politowski (1986). It attempts to capture the full information available by all the given features.

We have run several randomly generated examples in the domain of 8-puzzle using a guaranteed underestimating heuristic function. From the solution paths obtained, we have computed the minimum element of the h^*set corresponding to each feature vector. This

Table 2. Results of running A* with minimum of h^*set for various feature sets.

Features used	Manhattan	Manhattan misplaced tiles	Manhattan sequence count	Manhattan misplaced tiles sequence count
% Nodes expanded	100	81	77	42

value is then used in the second phase as the heuristic estimate of a node in algorithm A*. 10000 different random examples were then run with this new heuristic estimate. We observed that the number of node expansions was reduced from that of A* with Manhattan distance heuristics. All the solutions obtained were optimum. The results of running the algorithm after learning with different feature sets are presented in table 2. The table illustrates how the combination of different features drastically reduce node expansions. Note that though Misplaced tiles heuristics is dominated by the Manhattan estimate, using both the estimates together gives rise to significant improvement in search performance.

4.2 *Using h^*set as information*

Our objective is to find out if other stronger forms of information can be associated with the classes, such that they can be effectively used for more effective searching. If a best first search algorithm uses a single value as information, *minh* is the best estimate that can be associated with a class to guarantee admissibility. However, if the set of feasible optimum cost values of the states comprising the class is associated with a class, then stronger reinforcement to the heuristic estimate of a node can be made by using the path information. In this section we present algorithm CS* that makes use of the information of h^*set to make search more effective.

4.2a *Some definitions:* If the h^*set is known for each class, the possible values of optimum solution costs of the problem that can be obtained by extending the current path is given by $fset$ which represents the set of feasible values of the optimum solution cost of a path that is constrained to pass through n with a g value equal to the cost of the current pointer path.

DEFINITION 3
The $fset$ of a node n, given the pointer path P from s to n, and the value of h^*set at the node (i.e., $h^*set(hvec(n))$), is given by

$$fset(n, P) = \{x \mid x = g(n) + y \ \ \forall y \in h^*set(hvec(n))\}.$$

DEFINITION 4
We define the *minf* value of a node n, $minf(n, P)$ to be the minimum element of the corresponding $fset$.

$$minf(n, P) = min\,\{(fset(n, P))\}.$$

We make use of the following property : if C^* is the optimum solution cost from node s to a node γ, then if a node n lies on any optimum solution path from s to γ, the sum of the optimum cost from start node s to the node n and that from node n to a goal node γ equals C^*. This implies that the intersection of the *fset* of nodes lying on an optimum solution path must be non-empty. We introduce the notion of *pathint* which has a parallel in the concept of *pathmax* (Mero 1984; Dechter & Pearl 1985) in best first algorithms using single valued estimate.

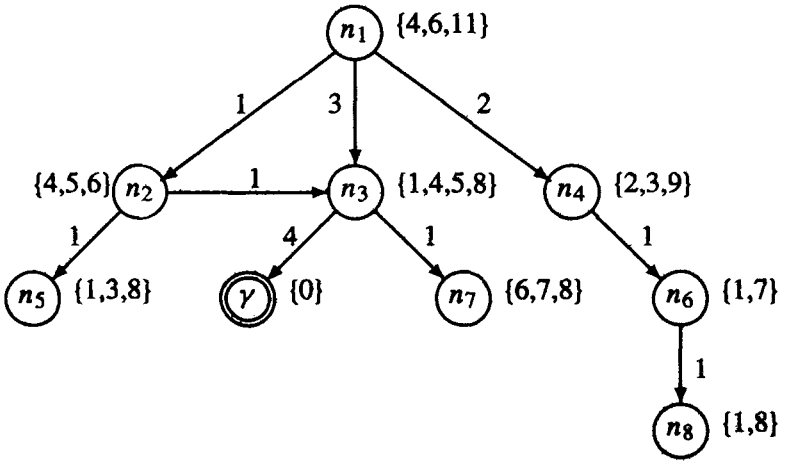
DEFINITION 5
The *pathint* of a node n along a pointer path $P = (n_0, n_1, \ldots, n_k)$, is defined to be the intersection of the *fsets* of all nodes belonging to the path. It is denoted by $Pset(n, P)$.

$$Pset(n, P) = \bigcap_{i=0}^{k} fset(n_i, (n_0, n_1, \ldots, n_i)).$$

This means that if C^* is the optimum solution cost, it will be a member of the *pathint* of all nodes on any optimal cost solution path. Retaining the *Pset* at every expanded node results in better selection of nodes for expansion than what is provided by A*.

DEFINITION 6
We define the *minP* value of a node n along a path P, $minP(n, P)$ to be the minimum element of its *Pset*.

Example 3. The above definitions are illustrated below with reference to the example graph in figure 1.



Note: The set shown against each node denotes the value of h^*set at the node.

Figure 1. A search graph.

$$
\begin{array}{llll}
fset(n_1, (n_1)) = \{4, 6, 11\} & Pset(n_1, (n_1)) = \{4, 6, 11\} \\
fset(n_3, (n_1, n_3)) = \{4, 7, 8, 11\} & Pset(n_3, (n_1, n_3)) = \{4, 11\} \\
fset(n_2, (n_1, n_2)) = \{5, 6, 7\} & Pset(n_2, (n_1, n_2)) = \{6\} \\
fset(n_3, (n_1, n_2, n_3)) = \{3, 6, 7, 10\} & Pset(n_3, (n_1, n_2, n_3)) = \{6\} \\
fset(n_5, (n_1, n_2, n_5)) = \{3, 5, 10\} & Pset(n_5, (n_1, n_2, n_5)) = \phi
\end{array}
$$

$$
minf(n_2, (n_1, n_2)) = 5 \qquad minP(n_2, (n_1, n_2)) = 6 \qquad \square
$$

DEFINITION 7

The upper bound to the solution cost obtained at node n denoted by $ub(n, P)$ is the maximum cost of an optimum solution path constrained to be an extension of path P. At any point during the search process, when the set of nodes S have been expanded, *minub* is defined to be the minimum of the upper bound of the nodes.

$$minub = min_{n \in S}\{ub(n, PP_{s-n})\}.$$

In this section, we outline a best first search algorithm which uses the information about the h^*set to direct its search process. We show that the use of *pathint* with h^*set provides a more informed and stronger pruning criterion than *pathmax* using a single valued estimate. We outline algorithm CS* that uses h^*set as state information and *pathint* for path information. It is guaranteed not to be worse than algorithm **A*** using similar information. We also present the iterative deepening version of the algorithm, CIDS* . In addition to making use of *pathint* to convey path information, it uses *front-union* to transfer additional information from one iteration to the next.

4.2b *Algorithm* **CS***: *Observation 1.* If an extension of the path P_{s-n^*} is an optimum solution from s to a goal γ, then

$$C^* \in fset(n^*, P_{s-n^*}).$$

Suppose node n_2 is a child of node n_1. A necessary condition that the arc $\langle n_1, n_2 \rangle$ is included in an optimum cost solution path is that $C^* \in fset(n_1, PP_{s-n_1})$ and $C^* \in fset(n_2, PP_{s-n_2})$, which is possible only if the intersection of these two fsets is non-empty, because, we must have

$$C^* \in fset(n_1, PP_{s-n_1}) \bigcap fset(n_2, PP_{s-n_2}).$$

When a node n_k is generated, let PP_{s-n_k} be the pointer path from s to n_k. We note that if an optimum solution path can be reached by expanding the node n_k along this path, then the optimum solution cost C^* must be a member of all nodes included in the pointer path. This leads to the following observation.

Observation 2. If the *pathint* at a node becomes empty, that branch of the search tree can be pruned as it is guaranteed not to lead to an optimum solution.

We may therefore consider retaining the *pathint* information at every node. The concept of *pathint* corresponds to the concept of *pathmax* (Dechter & Pearl 1985) in traditional A*

search using a single estimate at a node, but provides a much stronger pruning criterion. It enables us to remove from consideration those values of feasible optimum solution costs of the path such that there exists at least one node along the path whose set of feasible optimum solution costs does not contain that particular cost.

Observation 3. The *minP* value of a node can be effectively used as the estimate of a node during the node selection process.

We outline the algorithm CS* below which is a best first algorithm that uses *minP* as the heuristic estimate at a node.

Algorithm
With every node we associate the following information:

$$\langle n, fset(n, P), Pset(n, P), parent \rangle$$

1. [INITIALIZE:] minub = MAXh^* ;; The maximum cost of a solution in the domain.
OPEN $\leftarrow \langle s, fset(s, (s)), Pset(s, (s)) \rangle$ where $fset(s, (s)) = Pset(s, (s)) = h^*set(hvec(s))$
CLOSED $\leftarrow \phi$

LOOP:

2. [TERMINATE:] If OPEN is empty, exit with failure.

3. [SELECT:] Select the node from OPEN with minimum value for $minP(n, P_{s-n})$

4. [CHECK FOR GOAL:] If n is a goal node, exit with solution, $P P_{s-n}$.

5. [EXPAND and PRUNE:] Expand node n, generating the set M of its successors, and attach to them pointers back to n.

For all $m \in M$,
If $m \notin$ OPEN or CLOSED
add the tuple $\langle m, fset(m, P P_{s-m}), Pset(m, P P_{s-m}) \rangle$ to OPEN if $Pset() \neq \phi$.

If $m \in$ OPEN or CLOSED
direct its pointers along the path of lowest value of $g(m)$.

If m required pointer adjustment and was on CLOSED, reopen it.

If $max\{fset(m, P P_{s-m})\} < minub$,
set $minub = max\{fset(m, P P_{s-m})\}$
Remove from OPEN those nodes whose lower bound > *minub*

6. [CONTINUE:] Go to LOOP.

Computation of $Pset(n, P P_{s-n})$: If n_i and n_j lie on an optimum path from s to g,

$$f^* \in fset(n_i, P P_{s-n_i}) \cap fset(n_j, P P_{s-n_j}).$$

The *pathint* for a path is computed incrementally as follows:

$$Pset(s, (s)) = fset(s, (s)),$$
$$Pset(n, P) = Pset(m, P') \cap fset(n, P)\ If\ P = (P' \mid n).$$

Example 4. We will illustrate the algorithm by showing how it works for the example graph of figure 1. $s = n_1$ is initially on OPEN. It is put on CLOSED, and its successors $n_2(Pset() = (6))$, $n_3(Pset() = (4,11))$ and $n_4(Pset() = (4,11))$ are generated and put in OPEN. At this point, minub = 7 (from node n_2).

n_3 is first selected for expansion($minP(n_3, (n_1, n_3)) = 4$). Its successors are $\gamma(Pset() = \phi)$ and $n_7(Pset() = (11))$. γ is not put in OPEN because its *Pset* is empty and n_7 is not put in OPEN because its $minf$ value(10) exceeds the value of $minub(7)$.

n_4 is next expanded with a $minP$ value of 4, and n_6 is generated with $Pset = (4)$. Its $minP$ value is 4, and it is the next candidate for expansion. The successor n_8 is pruned because its *Pset* is empty. Now n_2 is expanded. n_5 is pruned because its *Pset* is empty, and n_3 is put in OPEN with $Pset = \{6\}$. n_3 is now expanded. n_7 is pruned(empty *Pset*), and γ is put in OPEN. γ is next selected for expansion, found to be a goal node, and the solution path $\langle n_1, n_2, n_3, \gamma \rangle$ is output.

To summarize, nodes are expanded in this order: $n_1, n_3, n_4, n_6, n_2, n_3, \gamma$. It may be observed that the nodes expanded by the algorithm A* using *minf* as the evaluation function at a node are $n_1, n_4, n_6, n_3, n_2, n_5, n_3, n_8, \gamma$. □

4.2c *Properties of CS**:

Lemma 1. If a node n expanded by CS* lies on an optimum path from s to a goal node* $\gamma \in \Gamma$, *then* $C^* \in fset(n^*)$.

Proof. The result follows from observation 1. □

Lemma 2. The value of minub provides a global upper bound to the optimum solution cost value C for the given problem.*

Proof. $minub = max\{fset(n, P)\}$ for some node n, and some path P_{s-n}. Therefore, $minub = g(n, P) + x$ where $x = max\{h^*set(hvec(n))\}$. A finite value of x implies that a finite solution path must exist from node n to a goal node γ with optimum solution cost $\leq x$. Since we have already found a path of cost $g(n, P)$ from s to n, there must exist a solution path from s to γ of cost $\leq g(n, P) + x = minub$. □

The value of *minub* can be used to restrict the *Pset* value of a node by removing from consideration those values that are greater than *minub*.

COROLLARY 1
If P is a path from s to n, and $minP(n, P) > minub$, *P cannot be part of an optimum cost path from s to goal* γ.

While this has no effect on the number of nodes expanded, this property enables us to prune from the OPEN set.

Lemma 3. At any time before CS terminates, there exists on OPEN a node n_i which is on an optimum path $P^*_{s-\gamma}$,* s.t. $C^* \in Pset(n_i, P_{s-n_i})$.

Proof. The proof is similar to that of A*(Nilsson 1980). □

COROLLARY 2
If n_i is the shallowest node belonging to an optimum solution path $P^*_{s-n'}$, then $g(n_i) = g^*(n_i)$.

Note. Note that our evaluation function at a node is path dependent, and the same node may be expanded more than once. However it is ensured that the *minP* values of the expanded nodes are non-decreasing.

Theorem 1. *Algorithm* **CS*** *is admissible.*

Proof. Suppose the algorithm terminates with goal node $\gamma \in \Gamma$ and solution cost $g(\gamma) = minP(\gamma, PP_{s-\gamma}) > C^*$. When t was chosen for expansion,

$$minP(t, PP_{s-t}) \leq minP(n, PP_{s-n}) \quad \forall n \in \text{OPEN}.$$

Therefore immediately prior to termination, all nodes on OPEN satisfied the condition

$$minP(n, PP_{s-n}) \geq minP(t, PP_{s-t}) > C^*.$$

But according to the previous lemma, there must have existed on OPEN, just before termination, at least one node n' on an optimum path with $C^* \in Pset(n', PP_{s-n'})$ *i.e.*, $minP(n', P) \leq C^*$. This is a contradiction. Hence algorithm CS* must terminate with an optimum solution. Therefore it is admissible. □

The *minP* value of a node obtained by using *pathint* can be effectively used as the estimate of a node during the node selection process and provides a stronger estimate than *minf* that makes use of *pathmax*.

Lemma 4. CS(h) never expands more nodes than algorithm* **A*(h)** *that uses minf as the heuristic function.*

Proof. For any node, n, selected for expansion by CS* , $minP(n, P) \leq f^*$ if n lies on an optimum solution to the goal along path P.
Because $Pset(n, P) \subseteq fset(n, P)$, $minf(n, P) \leq minP(n, P) \leq f^*$ A*(h) expands all nodes n such that $f(n) < c^* = f^*$
CS* (h) does not expand any node n such that $f(n) > c^*$.
Also, if A* and CS* use the same tie-breaking rule, CS* does not expand any more nodes whose $f(node) = c^*$ than A* does.
In fact A* is a special case of CS* such that A*$(h) \equiv$ CS*(h') where $h^*set_{h'}(n)$ is the continuous range from $h(n)$ to ∞. □

In some cases CS* expands less nodes than A* The extra pruning in CS* can be attributed to these two factors:

1. By using *pathint*, it takes the intersection of the possible values of f for every node along a path. For a good set of features h, the *pathint* for off-track nodes may be empty. This results in pruning of these paths.
2. By using a global Upper Bound on the value of f^*, it is possible to further tighten the value of *pathint* at nodes, which is effective in reducing the size of the OPEN set.

Size of OPEN: Let $\mathcal{F}$ be the frontier set of the implicit search tree generated by A*, i.e., the set of all tip nodes. In the case of A*, OPEN contains all successors of $\mathcal{F}$ that are not members of $\mathcal{F}$. In CS* , OPEN contains only those successors of $\mathcal{F}$ s.t. $Pset(n, P) \neq \phi$ and $minP(n, P) < minub$.

Also we have proved that any node expanded by CS* is also expanded by A*. This means that the largest size of the OPEN set for algorithm CS* cannot be more than that of A*, but it can be less.

4.2d *When is a heuristic determiner more powerful than another?* In the general case, it is difficult to evaluate the relative merits of two heuristic determiners, $\mathbf{h}_1$ and $\mathbf{h}_2$. Loosely speaking, $\mathbf{h}_1$ is better than $\mathbf{h}_2$ if $\mathbf{h}_1$ discriminates between on-track and off-track nodes more effectively. However it is possible to make some simple observations:

Lemma 5. A heuristic determiner $\mathbf{h}_1$ *dominates a heuristic determiner* $\mathbf{h}_2$ *if*

$$\forall \mathbf{k}\ h^*set_{h_1}(\mathbf{k}) \subseteq h^*set_{h_2}(\mathbf{h}_2)$$

if $h^*set_{h_i}$ *denotes the* h^*set *when using* h_i *as the feature set.*

We have assumed that the cardinality of h^*set is finite and reasonable so that it is possible to store the set directly. If h^* assumes real values, it may be the case that the range of h^*set is continuous. In this case we can just keep the lower bound and upper bound of the range. This is equivalent to using the lower bound (the value of *minf*) as the heuristic estimate and using the upper bound to compute *minub*. A suitable representation can also be designed if the range of h^*set is piecewise continuous.

4.2e *Experimental results:* In table 3 we show how running of the CS* algorithm using the information of h^*set corresponding to each feature vector value can further reduce node expansions. The results can be compared with that of table 2.

4.3 *CIDS* – Iterative deepening version*

Consider the iterative deepening version of the above algorithm. In an iterative deepening algorithm, a threshold is selected for the current iteration and depth first search is executed until the estimates of the frontier nodes cross the threshold. For an iterative deepening algorithm, at the end of any iteration, the OPEN list defines the frontier of the implicit

Table 3. Results of running CS* with *h*set* for various feature sets.

Features used	Manhattan	Manhattan misplaced tiles	Manhattan sequence count	Manhattan misplaced tiles sequence count
% Nodes expanded	100	75	69	35
Number of classes	25	78	168	403

search tree generated, which consists of all the nodes that have been generated, but not yet expanded.

DEFINITION 8
We define the *frontier set* at the end of the ith iteration of an iterative deepening algorithm with cutoff f' to be the set of nodes $\mathcal{F}$, such that the *minP* values of these nodes exceeds the threshold for the current iteration (f'), while the *minP* value of their parents lie within the threshold.

Thus the set $\mathcal{F}$ defines a cut such that the start node s lies on one side, and any goal γ, if it exists, lies on the other side of the cut. If there exists a solution to the problem, any optimum path to the solution must pass through at least one node included in the cut. Also, when the shallowest such node, m, was expanded, the g-value of the node was $g^*(m)$ (by corollary 2). Therefore the *Pset* value of the node m will include the optimum solution cost. This motivates us to define the concept of *front-union*.

DEFINITION 9
The *front-union* of the ith iteration of the iterative deepening CS* algorithm is defined to be the set of f values in the union of the *pathint* values of all nodes defining the cut.

$$Funion(\mathcal{F}^i) = \bigcup_{n \in \mathcal{F}^i} Pset(n, PP_{s-n}).$$

CIDS* is an iterative deepening algorithm that makes use of *front-union*. It is an extension of the IDA* algorithm, with *pathint* calculation and pruning by *minub* like that of CS* . At the end of an iteration, the *front-union* is computed. This value is backed up as the *Pset* of the start node s at the beginning of the next iteration. This enables the wisdom gained in an iteration to be propagated to subsequent iterations. We present our justification of the algorithm below:

A very important property of *front-union* is that if there exists a solution to the problem, the front-union must contain the optimum solution cost.

Lemma 6. If there exists a path from s to a goal $\gamma \in \Gamma$, *and a goal has not yet been found after the* ith *iteration of CIDS*, the optimum solution cost* C^* *must be an element of the ith front-union set* $\mathcal{F}_i$.

Proof. We have noted that $\mathcal{F}_i$ defines a cut in the search space. If there exists a path from s to a goal node $\gamma \in \Gamma$, the *ith* front-union set must intercept one of the nodes on the optimum solution path. Let that node be m. We note that when the node m was selected for expansion, the optimum solution path to the node m has been found. Therefore $g(m, PP_{s-m}) = g^*(m)$. $h^*(m) \in h^*set(hvec(m))$ by definition of h^*set. Therefore $f^*(m) \in fset(m, PP_{s-m})$. Since an extension of PP_{s-m} is an optimum cost path, $f^*(m) \in Pset(m, PP_{s-m})$. Therefore $C^* \in Pset(m, PP_{s-m})$.

$$\exists m \in \mathcal{F}_i \text{ s.t. } C^* \in Pset(m, PP_{s-m}).$$
$$\textit{Therefore, } C^* \in Funion(\mathcal{F}_i).$$

□

The above lemma implies that, for subsequent iterations, a cost value which is not included in $Funion(\mathcal{F}_i)$ need not be considered. This means that for the next iteration, we can set

$$Pset^{i+1}(s, \langle s \rangle) = Pset^i(s, \langle s \rangle) \cap Funion(\mathcal{F}^i)$$

But $Funion(\mathcal{F}^i) \subseteq Pset^i(s)$. Therefore

$$Pset^{i+1}(s) = Funion(\mathcal{F}^i)$$

Example 5. We show how the algorithm CIDS* works for the example graph of figure 1.

- At the beginning of the first iteration, $Funion = (4,6,11)$, threshold $= 4$. Nodes expanded are

  1. n_1, path $= (n_1)$, $Pset = (4, 6, 11)$,
  2. n_3, path $= (n_1, n_3)$, $Pset = (4, 11)$,
  3. n_4, path $= (n_1, n_4)$, $Pset = (4, 11)$,
  4. n_5, path $= (n_1, n_4, n_5)$, $Pset = (4, 11)$.

  The nodes that define the frontier at the end of the 1st iteration are:

  1. n_2, path $= n_1, n_2$, $Pset = (6)$,
  2. γ, path $= n_1, n_3, \gamma$, $Pset = \phi$,
  3. n_7, path $= n_1, n_3, n_7$, $Pset = \phi$,
  4. n_8, path $= n_1, n_4, n_6, n_8$, $Pset = \phi$.

  Therefore $\mathcal{F}^1 = \{n_2, \gamma, n_7, n_8\}$ and $Funion(\mathcal{F}^1) = (6)$.

- At the beginning of the second iteration, $Funion = (6)$.
  Threshold $= min\{minP(n)\}$ $\forall n \in Funion(\mathcal{F}_1) = 6$.

  Nodes expanded are

  1. n_1, path $= n_1$, $Pset = (4, 6, 11)$,
  2. n_2, path $= (n_1, n_2)$, $Pset = (6)$,
  3. n_3, path $= (n_1, n_2, n_3)$, $Pset = (6)$,
  4. γ, path $= n_1, n_2, n_3, \gamma$, $Pset = (6)$.

Table 4. Nodes expanded by CIDS* in the example of figure 1.

Iteration #	Threshold	Nodes expanded by CIDS*	$Funion(\mathcal{F}_i)$
1	4	n_1, n_3, n_4, n_6	{6}
2	6	n_1, n_2, n_3, γ	

The maximum nodes that can be generated are:

1. n_5, path $= (n_1, n_2, n_5)$, $Pset = \phi$,
2. n_7, path $= (n_1, n_2, n_3, n_7)$, $Pset = \phi$,
3. n_3, path $= (n_1, n_3)$, $Pset = \phi$,
4. n_4, path $= (n_1, n_4)$, $Pset = \phi$.

In table 4 we show the nodes expanded by algorithm CIDS* for the example in figure 1. For comparison we have also presented the nodes expanded by algorithm IDA* which makes use of *minf* as the evaluation function at a node in table 5. We note that CIDS* expands nodes n_4 and n_6 in the first iteration, but they are not expanded again in the second iteration. □

The concept of *front-union* is often useful in being able to restrict the *pathint* set of the start node from information that is deeper down the tree and hence expected to be more accurate. This may prevent some nodes that are expanded in one iteration from being expanded/generated in subsequent iterations, as illustrated by the above example.

5. δ-Risk algorithm using fuzzy set model

The algorithm presented above is not robust in the presence of incomplete information. Only a completely error-free heuristic determiner can guarantee termination with an optimum solution. In our learning scheme, this cannot be ensured except in the limit. Also, in order to guarantee admissibility, the algorithm uses an over-cautious approach in selecting nodes for expansion. In the learning phase, it is possible to acquire statistics about the distribution of the values of h^*set . This information can then be fruitfully used to make a better decision about which node to expand next at the risk of missing the optimum solution sometimes. However the risk factor can be fed as a parameter to the algorithm.

Table 5. Nodes expanded by IDA* in the example of figure 1.

Iteration #	Threshold	Nodes expanded by IDA*
1	4	n_1, n_3, n_4, n_6
2	5	$n_1, n_3, n_4, n_6, n_8, n_2, n_3, n_5$
3	6	$n_1, n_3, n_4, n_6, n_8, n_2, n_3, n_5, \gamma$

We find that if we insist on solving problems faster on the average while using the same features, we have to compromise on the solution quality obtained. However, we usually like to have some control over the deterioration of the solution quality. We describe a scheme where this poorness of quality can be parameterized so that we have a handle on the quality of the solution cost. In addition to learning the minimum value of h^*set corresponding to every feature vector value, we may learn all the elements of h^*set and their distribution while solving problems. A typical h^*set distribution in the domain of 8-puzzle is shown below.

h_1	h_2	h_3	h^*set distribution						
			frequency	0.024	0.138	0.289	0.407	0.119	0.020	0.003
16	7	16								
			h^* value	16	18	20	22	24	26	28

We augment the model outlined previously by keeping the distribution of h^*set. The results of the previous algorithms are still valid for these algorithms.

There have been several algorithms aimed at finding solutions faster with controlled reduction in solution quality. Pohl (1970) uses an evaluation function $f_\varepsilon = (1-\varepsilon)g + \varepsilon h$, for $0 \leq \varepsilon < 1$. For values of $\varepsilon > 1/2$, we may expect solutions with less search effort in certain cases. Pearl (1984) has outlined algorithm R^*_δ that works with limited risk using information about the uncertainty of the heuristic function. The uncertainty information of the h values at a node has been modelled as a probability distribution function at a node. He has proposed several alternatives for selecting the node to expand given the value of δ, and shown that these algorithms are δ-risk admissible. Bramanti-Gregor & Davis (1993), propose a statistical method of combination of features. The method provides a probabilistic estimate of the upper-bound to the solution error by calculating the certainty bounds of the supremums of the standard errors of the predicted values. Bramanti-Gregor & Davis (1992) evaluate the performance of this method using some methods aimed at producing near-optimal solutions with reduced node expansions.

We model similar concepts as Pearl (1984); but since we are working with sets, the possibilistic model (Klir & Folger 1988) seems more appropriate in this situation. In the standard probabilistic search algorithms in literature, a necessary condition for the algorithms to work well is that the h^*set has elements that are fairly concentrated around a central function (Chenoweth & Davis 1991). In our model, that is not a necessary criterion — a discriminatory set of features that can effectively discriminate between off-track and on-track nodes by applying the concept of *pathint* will work well. In the last section, we had modelled the h^*set corresponding to each value of the feature vector, as well as the *fset* and the *Pset* as 'crisp sets'. If all members of the h^*set are not equally likely, we may use the information about the likelihood of the individual members of the h^*set to make a more informed decision provided that we relax the optimality requirement. We propose to assign 'membership grades' to each element of the h^*set proportional to the frequency of their past occurrences. We take this as a measure of the possibility of their future occurrence. Thus the h^*set can be modelled as a fuzzy set. We modify the algorithm CS* to work with fuzzy sets. *min* is taken as the intersection operation, and *max* as the union operation. Selection of a node for expansion is by taking the least of the f_δ value of

the *Pset* for each set in OPEN, where δ is our confidence parameter. This means that we initially ignore those values of likely f^*s whose possibility of occurring is less than δ.

Before outlining the algorithm, we give a few definitions.

DEFINITION 10

The distribution of h^*set at a node (denoted by $Dh^*set(n)$) is the normalized [4] possibility distribution of feasible h^* values corresponding to the feature vector value at the node ($hvec(n)$).

$C_{h^*}(x, n)$ is the distribution of the exact/relative number of times that the value of h^* was x for nodes with feature vector value $hvec(n)$.

$\rho_{h^*}(x, n)$ is the Possibility Density Function corresponding to $DH^*set(n)$ that measures the relative possibility that the value of h^* is x corresponding to the feature vector value $hvec(n)$.

DEFINITION 11

Dfset(n, P): The distribution of fset at a node (n) along a path P_{s-n} from start to node n is the normalized possibility distribution of feasible f^* values corresponding to the feature vector value associated with the node ($hvec(n)$) and the g value corresponding to the current path at the node.

$C_f(x, n)$ is the distribution of the count of the f-value of a node induced by $C_{h^*}(x, n)$,

$\rho_f(x, n)$ is a density function corresponding to *Dfset*. It measures the relative possibility that the value of the optimum f value through the node is x.

$$\rho_f(y, n) = \rho_{h^*}(y - g(n), n).$$

DEFINITION 12

DPset(n, P) is the possibility distribution function of the value of optimum solution that is an extension of path P.

$$DPset(s, (s)) = Dfset(s)$$
$$DPset(n, P \mid m, n) = DPset(m, P \mid m) \cap Dfset(n)$$

min is taken to be the intersection operation.

$\rho_{Pset}(x, P)$ is the possibility that the value of the optimum solution through the path is x.

$$\rho_{Pset}(x, P) = \bigcap_{n \in P} \rho_f(x, n)$$
$$= min_{n \in P}\{\rho_f(x, n)\}.$$

DEFINITION 13

The f_δ value of a fuzzy set is defined to be the minimum element of the α-cut of the given

[4] A possibility distribution is said to be normalized if the highest membership grade value is 1. To normalize an arbitrary possibility distribution, we find the peak(s) of the possibility density function, and then scale the distribution such that the maximum peak becomes 1.

set with $\alpha = \delta$. Let A be a fuzzy set, A_α be the α-cut of A.

$$A = \sum_{i=1}^{n} \rho_i / x_i,$$
$$A_\alpha = \{x \in X \mid \rho_a(x) \geq \alpha\},$$
$$f_\delta = \{x \mid x \in A_\alpha,\ x \leq y\ \forall y \in A_\alpha\}.$$

5.1 *Learning phase*

A set of random problems are generated according to the given distribution, and corresponding to the optimal solutions to these problems, the $C_{h^*}(x, \mathbf{h})$ values are updated for all sets of feature vectors $\mathbf{h}$. We use $C_{h^*}(x, \mathbf{h})$ values in the problem solving phase.

Computation of $Dh^*set(\mathbf{h})$ values: If R is the set of all feasible values of $h^*set(\mathbf{h})$, and $|R| = K$, in the absence of any examples, we let the possibility at each point in the feasible range be $1/K$. After encountering N examples for nodes whose feature vector value is $\mathbf{h}$, we update the value of $C_{h^*}(x, \mathbf{h})$ as follows: $C_{h^*}(x, \mathbf{h}) = n_x$ where n_x is the number of times that the value of h^* was x corresponding to a state having feature vector value $= \mathbf{h}$. Then we compute the distribution as follows following Hansson & Mayer (1989).

$$Dh^*set(x, \mathbf{h}) = \frac{n_x + 1}{X + 1} \quad where \quad X = max_x\{n_x\}.$$

5.2 *Problem solving phase*

If p_1 is the possibility of having an optimum solution cost of C through node n_1, and p_2 is the possibility of having an optimum solution cost of C through node n_2, child of n_1, then the possibility of having an optimum solution cost of C including the arc (n_1, n_2) is given by $min(p_1, p_2)$. If D_1 and D_2 be the distributions of the fsets of the parent and the child respectively, then the distribution of the possible optimum solution costs through the arc is given by $D_1 \cap D_2$ where $\cap$ uses the *min* operator to compose individual elements of the distribution. If P is the current path terminated at node n, then the possibility distribution of optimum solution cost values given the path information can be obtained by

$$\bigcap_{m \in P} Dfset(m, PP_{s-m}),$$

which is obtained by the possibility density function given by

$$\rho_P(C, n) = min_{m \in P}\{\rho_f(C, m)\}.$$

To guarantee optimum solution we have to select for expansion that node from OPEN s.t. the minimum non-zero f-value of the *pathint* of the node is minimum among all the nodes in OPEN. If we are prepared to trade off the optimality requirement partially in the hope of quickly reaching a solution, we may ignore those members of the distribution whose possibility values lie below a certain threshold. This motivates us to choose a node for expansion that has the minimum 'significant' f-value. The threshold of significance

is controlled by the parameter δ. δ is a measure of the risk associated with missing the optimum solution, and is called the confidence parameter. If no solution is found with parameter δ, the value of δ is reduced and the algorithm rerun. This continues until a solution is obtained.

Algorithm FS^*_δ

1. [INITIALIZE:] minub = ∞ OPEN $\leftarrow$ $\langle s, Dfset(s, (s)), DPset(s, (s))\rangle$ where $Dfset(s, (s)) = DPset(s, (s)) = Dh^*set(s)$. CLOSED $\leftarrow \phi$, $delta = \delta$

LOOP:

2. [TERMINATE:] If OPEN is empty, exit with failure.

3. [SELECT:] For all nodes $m \in$ OPEN, Select the node for expansion which has the minimum value for $f_\delta(m)$ out of all nodes in OPEN. If no such node exists, set $delta = delta/2$ and repeat step 3.
Call the selected node n.

4. [CHECK FOR GOAL:] If n is a goal node, exit with solution.

5. [EXPAND and PRUNE:] Expand node n, generating the set M of its successors.

For all $m \in M$,
If $m \notin$ OPEN or CLOSED
add the tuple $\langle m, Dfset(m, P), DPset(m, P)\rangle$ to OPEN if $Pset(m) \neq \phi$.

If $m \in$ OPEN or CLOSED
direct its pointers along the path of lowest value of $g(m)$.

If m required pointer adjustment and was on CLOSED, reopen it.

If $max\{fset(m)\} < minub$,
set $minub = max\{fset(m)\}$
Remove from OPEN those nodes whose lower bound > $minub$

6. [CONTINUE:] Go to LOOP.

Computation of the pathint *value at a node*:
If $\rho_{Pset}(y, parent) = r_1$,
$\rho_f(y, child) = r_2$,
then $\rho_{Pset}(y, child) = min(r_1, r_2)$.

FS^*_δ selects that node for expansion which has minimum value for $f_\delta(n)$. δ is defined to be our confidence parameter. It is a measure of the risk associated with missing a solution. Suppose our algorithm FS^*_δ has found a solution C with some node m in OPEN, s.t. $f_0(m) < C$. What is the possibility that we would have found a solution with cost $< C$ by expanding m? Our knowledge about the cost of optimum solutions containing the path

P_i,($\langle s \ldots m \rangle$) is represented by $DPset(m, PP_{s-m})$. Therefore the given possibility can be computed as

$$max\{p|p = \rho(x, m) \ \& \ x < C\}.$$

If $C* \leq f_\delta(n)$, this possibility is $< \delta$. In other words, the possibility of the current solution being an optimum solution is $> (1 - \delta)$.

5.3 *Properties of FS**

Observation 4. CS* is a special case of FS^*_δ with δ equal to a small positive value. When the scheme of computation of *Dh*set* is as above, FS^*_δ with $\delta = 0$ is equivalent to A*(h) where h is the underestimating heuristic used in assigning the values of *Dh*set*.

*Lemma 7. The algorithm is robust and can be always made to terminate with a solution even if the h*set is not complete.*

Proof. While assigning the possibility distribution we allow for sufficient noise tolerance and assign some possibility value, however small, to values of h^* that have even the remotest chance of being included in the set. Under such condition, there will always exist some positive value of δ for which the algorithm will find a solution. Since the algorithm tries with progressive reductions in the value of δ, termination with a solution is ensured. □

Table 6. Performance of algorithm FS* for different amounts of learning and different values of δ versus the performance of A*.
NE : Average relative percentage of nodes expanded (compared to A*(h_1).
SO : Average relative percentage of suboptimum solutions found (compared to A*(h_1).

δ	FS*(h_1, h_2)		FS*(h_1, h_3)		FS*(h_1, h_2, h_3)	
	Ave NE	Ave SO	Ave NE	Ave SO	Ave NE	Ave SO
0.1	75.6	0.3	75.9	0.1	39.5	0.2
0.2	74.3	1.4	75.8	0.1	37.5	0.5
0.3	71.4	3.0	73.8	1.0	37.4	0.6
0.4	69.3	3.0	70.1	2.1	35.3	2.8
0.5	64.7	5.1	68.2	3.9	33.8	4.1
0.6			58.0	10.8	32.7	6.2
0.7	60.5	8.0	50.6	13.5	31.2	8.9
0.8	53.5	13.9	48.2	15.9	30.0	12.0
0.9	53.3	15.6	42.0	18.6	27.1	17.1
1.0	53.1	20.2	39.1	23.0	25.8	17.5

6. Conclusion

This paper addresses the issue of learning in admissible search. The information requirement for best first search has been studied. For this, we have distinguished the domain information as given by the feature values at the nodes from the heuristic determiner. The latter is learned from the former, and both can be of quite general form, the goal being to design a powerful determiner for a given domain, that can discriminate effectively between on-track and off-track nodes. The features can be arbitrary, and we have assumed a tabular form for representation of the heuristic determiner. Algebraic formulae for combining the features give constant space representations, but a particular algebraic form may not be suitable to combine effectively a given set of features for a domain. Furthermore, they cannot serve as a model for powerful algorithms like CS* . However they may lead to large space requirement.

References

Bramanti-Gregor A, Davis H W 1992 Strengthening heuristics for lower cost optimal and near optimal solution in A* search In *Proceedings of 10th European Conference on Artificial Intelligence* (New York: John Wiley & Sons) pp 6–10

Bramanti-Gregor A, Davis H W 1993 The statistical learning of accurate heuristics. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pp 1079–1085

Chenoweth S V, Davis H W 1991 High-performance A* search using rapidly growing heuristics. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence* pp 198–203

Christensen J, Korf R E 1986 A unified theory of heuristic evaluation function learning. In *Proc. Natl. Conf. AAAI*

Dechter R, Pearl J 1985 Generalized best-first search strategies and the optimality of A*. *J. Assoc. Comput. Mach.* 32: 505–536

Hansson O, Mayer A 1989 Heuristic search as evidential reasoning. In *Proceedings of the Fifth Workshop on Uncertainty in AI*

Klir G J, Folger T A 1988 *Fuzzy sets, uncertainty, and information* (Englewood Cliffs, NJ: Prentice Hall)

Lee K F, Mahajan S 1988 A pattern classification approach to heuristic function learning. *Artif. Intell.* 36: 1–25

Mero L 1984 A heuristic search algorithm with modifiable estimate. *Artif. Intell.* 23: 13–27

Nilsson N J 1980 *Principles of artificial intelligence* (Palo Alto, CA: Tioga)

Pearl J 1984 *Heuristics — Intelligent search strategies for computer problem solving* (Reading, MA: Addison–Wesley)

Pohl I 1970 First results on the effect of error in heuristic search. In *Machine intelligence* (American Elsevier) 5: 219–236

Politowski G 1986 *On the construction of heuristic functions*. Ph D thesis, University of California, Santa Cruz, CA

Samuel A L 1963 Some studies in machine learning using the game of checkers. In *Computers and thought* (eds) A Feigenbaum, J Feldman (New York: McGraw-Hill)

Samuel A L 1967 Some studies in machine learning using the game of checkers. ii — Recent progress. *IBM J. Res. Dev.* 11: 601–607