Practical and Theoretical Considerations in Heuristic Search Algorithms

Ira Pohl
Information Sciences
University of California at Santa Cruz

CONTENTS

1. The need for heuristics
   Discussion of the TSP (traveling salesman problem) as NP, implying that a satisfactorily efficient algorithm will rely heavily on heuristics. A simple NP reduction between the Hamilton path problem and the TSP problem will be proved.

2. A brief history of heuristic search programs
   Especial emphasis on the Michie-Nilsson-Pohl model of heuristic search. In the discussion of efficiency, a new condition of the monotone property will replace the Nilsson consistency assumption. It will be proved that this condition is sufficient for all purposes that consistency is used for, and in practice is easier to check.

3. Error analysis as a principal theoretical tool
   The importance of this concept. Principles of worst case error analysis will be applied to tree spaces. This discussion will extend to an analysis of bidirectional search. Outstanding research questions.

4. Practical application to a difficult problem
   The Held-Karp bounding function as a heuristic function for TSP. The principal of dynamic weighting in heuristic search.

5. Miscellany
   Knowledge representation in heuristic search programs, research questions, other topics...

*This research is supported by NSF grant DCR-74-20460 and was partially supported by ZWO of The Netherlands while the author was at the Vrije Universiteit, Informatica, Amsterdam.

55
THE NEED FOR HEURISTICS

The Cook (Cook, 1971) results on the hypothesized non-polynomial time complexity of an important class of combinatorial and logical computations— even where these are decidable problems—places a new emphasis on the need to understand heuristic techniques. What has long been a main theme of artificial intelligence—that many difficult problems can only be efficiently attacked by programs using ad-hoc rules systematized by experts knowledgeable in the problem domain—has now been given further backing by NP-theory (Aho, Hopcroft, and Ullman, 1974). It is almost universally accepted that heuristics will remain a vital method for the efficient solution of even finite decidable questions.

One such practically difficult problem is that of finding optimal solutions to the Traveling Salesman Problem (TSP). As it is a central question in operations research, it has been extensively investigated. The highly developed theory of this problem provides a rich semantics for use in computational solutions. Unlike many toy-problems, such as the Tower of Hanoi puzzle, which are used by artificial intelligence researchers to illustrate the power of their various problem solving systems, the TSP is a problem of known intransigence and importance.

**TSP symmetric case**

G = (X,E) is a finite undirected graph
c : E → R⁺ where each edge has a positive cost and
\|X\| = n there are n nodes.
Π(n) is the set of all permutations on the n nodes for
π₁(n) ∈ Π(n), π₁(n) = (i₁,i₂,...,iₙ) and

c(π(n)) = \sum_{j=1}^{n-1} c(i_j,i_{j+1}) + c(i_n,i_1)

is the cost of a tour represented by the permutation π₁(n). The TSP is to find a permutation π*₁(n) ∈ Π(n) with minimum cost.

![TSP diagram]

<table>
<thead>
<tr>
<th>edge</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1,2)</td>
<td>6</td>
</tr>
<tr>
<td>(1,3)</td>
<td>1</td>
</tr>
<tr>
<td>(1,4)</td>
<td>5</td>
</tr>
<tr>
<td>(1,5)</td>
<td>7</td>
</tr>
<tr>
<td>(2,3)</td>
<td>6</td>
</tr>
<tr>
<td>(2,4)</td>
<td>4</td>
</tr>
<tr>
<td>(2,5)</td>
<td>3</td>
</tr>
<tr>
<td>(3,4)</td>
<td>8</td>
</tr>
<tr>
<td>(3,5)</td>
<td>2</td>
</tr>
<tr>
<td>(4,5)</td>
<td>5</td>
</tr>
</tbody>
</table>

\[π^*₁(n) = (1,3,5,2,4)\]

\[c(π^*₁(n)) = 1+2+3+4+5 = 15\]
In (Karp, 1972), it is shown that the symmetric Hamilton circuit problem is an NP-problem. The Hamilton circuit problem is to find in a graph a simple circuit which traverses all the nodes of the graph. It is obvious that if there was a program which solved TSP in polynomial time, it could be used to solve the Hamilton circuit problem in polynomial time. What is more surprising is that given a program which on complete graphs finds a solution \( \pi'(n) \) to the TSP and \( c(\pi'(n)) \leq \alpha c(\pi^*(n)) \) where \( \alpha > 1 \) and \( \pi^*(n) \) the optimal tour then this program could be used to produce a polynomial time solution to the Hamilton circuit problem.

**Proof:**

Given \( G = \{X,E\} \) generate \( G' \) as follows:

if \((x_i,x_j) \in E\) then it is in \( G' \) with \( c(x_i,x_j) = 1 \)

if \((x_i,x_j) \notin E\) then it is in \( G' \) with \( c(x_i,x_j) = \alpha n \).

This recoding of \( G \) takes order \( n^2 \) steps.

Let \( P \) be a program acting on complete graphs and solving TSP to within \( \alpha \) times the optimal tour, then \( P(G') \) will generate some solution tour in polynomial time.

If there was a Hamilton circuit in \( G \), then there is a tour in \( G' \) of cost \( n \). Tours in \( G' \) which included edges not found in \( G \) cost at least \( n+1+\alpha \). Therefore when a Hamilton circuit exists in \( G \) it will be found by \( P(G') \) since only those tours are within \( \alpha \) of optimal cost. When no Hamilton circuit exists in \( G \) then all tours in \( G' \) will be at least cost \( \alpha n+n-1 \). Thus \( P(G') \) decides the question of Hamilton circuits in \( G \).

We can conclude from this that even finding algorithms which efficiently generate reasonable feasible solutions to the TSP in complete graphs will require heuristic programming methods. The list of problems which are known to be NP is already quite large and it is reasonable to conjecture that games such as chess or go, under a suitably devised generalization*, are also in this category.

**HEURISTIC SEARCH PROGRAMS**

Programs whose main emphasis is to search large spaces by the use of heuristics have been central to artificial intelligence research since its inception. The late 1940's and early 1950's devoted attention to the selective search of spaces defined by such games as chess and checkers (Shanno, 1950; Turing, 1953). The mid-1950's through the early 1960's saw the development of search methods which were general in that they could be applied to deductive problems from a variety of domains (Newell, Shaw and Simon, 1960).

*Chess has a finite though large number of legal board positions. In principal all these positions could be catalogued lost, won or drawn. To avoid this, one needs to invent an infinite number of such games which are chess-like but distinct. One possibility would be to make chess boards \( 8 \times n \) where \( n > 8 \).
PROBLEM-SOLVING AND DEDUCTION

The mid-1960's produced two striking contributions, both aimed at providing a uniform formal basis for discussing and implementing deductive search. The resolution principle and its associated search strategies (Robinson, 1965) provided one formal model and computational scheme which gained immediate hegemony over prior theorem proving models. The work on graph models of problem domains and the consequent rephrasing of deductive tasks into this language (Doran and Michie, 1966) provided an adequate uniform language to discuss and compare various heuristic search schemes.

So far the 1970's have seen two further themes elaborated; one has been a partial unification of the graph model and resolution model of search processes and a consequent crosstfertilization of useful theory and technique between both (Kowalski, 1972; Michie and Sibert, 1974). The second theme, which is the hallmark of progress in much of computer science, is the incorporation of heuristic search techniques as features of high-level programming languages (Hewitt and Smith, 1975).

These formalizations also make it possible to develop a theory of efficiency. While this has become the chief subject of research in theoretical computer science under the banner of complexity-theory, it is still more discussed than pursued in artificial intelligence. The mathematical ugliness of the domains investigated by artificial intelligence is a formidable barrier to obtaining results comparable to those of, for example, the sorting theorist (Pohl, 1975). Nevertheless important advances have occurred (Minsky and Papert, 1969) in artificial intelligence problem domains and heuristic search efficiency will form the chief topic of discussion in this paper. In heuristic search, these questions are deeply related to the accuracy of the heuristic functions. So that search efficiency is bound up with the analysis of error in the heuristics used for guiding search. In one sense the problem of knowledge representation by which I mean the conversion of human expertise into machine usable form, is reformulated as a problem of accuracy and efficiency. The program must have an appropriate representation of the problem and a search function which is accurate with respect to this representation. Either of these requirements by itself is not enough.

ON REPRESENTING PROBLEM DOMAINS
AND SEARCH ALGORITHMS

In order to discuss these issues more concretely, we make use of a graph model of heuristic search and a program which can search problems represented in this model.

A problem space is a locally finite directed graph $G$.

$$G = (X,E)$$

$X = \{x_1, x_2, \ldots \}$, $X$ is the set of nodes and can be infinite

$E = \{(x_i, x_j) \mid x_i, x_j \in X \text{ and } x_j \notin \Gamma(x_i)\}$, $E$ is the set of edges and can be infinite, however $\#(x_i)$ must be finite.
\( \Gamma \) is the successor mapping and if \( x \in \Gamma(y) \) then \\
y \in \Gamma^{-1}(x) \) the predecessor mapping

In using directed graphs to represent problem domains a state description is associated with each node. The successor mapping defines the structural character of the problem space, identifying for each node its immediate neighbors. A problem consists of some node (or set of nodes) which is the initial node and another node (or set of nodes) which is the terminal node. A solution to a particular problem is a path in \( G \) from the (an) initial node to the (a) terminal node. One thinks of a game such as the 15-puzzle where a state description represents the placement of pieces and the successor relation produces the legal moves.

An important sub-category of problems treated by this model associates positive cost with each edge. A solution is then called optimal if it is the least costly path. Frequently in combinatorial problems an optimal or near optimal solution is required.

\[
c : E \rightarrow \mathbb{R}^+ \text{ costs of the edges} \\
\mu(s,t) = (s = x_1, x_2, \ldots, x_k = t) \text{ is a path from } s \text{ to } t \text{ then} \\
c(\mu(s,t)) = \sum_{i=1}^{k-1} c(x_i, x_{i+1})
\]

It is sometimes of interest to treat \( c \) as a computational cost in producing a state. When this is the case the default understanding is that \( c \) is unity, i.e., the cardinality metric.

The algorithm that is presented here is the Heuristic Path Algorithm (HPA). It is one of a number of such algorithms (Nilsson, 1971; Michie and Ross, 1970; Slagle and Bursky, 1968); and is selected from this list because it is sufficiently general so that key issues of heuristic search can be illustrated in terms of it; and because the author and others have developed the work on error and efficiency in this schema (Pohl, 1970a).

**HPA description**

\( G = (X, E) \), the problem graph
\( c : E \rightarrow \mathbb{R}^+ \) = edge costs - most often 1
\( s = \text{start or initial node, } t = \text{terminal or goal node} \)
\( g(x) = \text{the cost of the path found by HPA from } s \text{ to } x \)
\( h(x) = \text{the estimated cost of a path from } x \text{ to } t \)
\( w(x) = \text{a weighting term, } 0 \leq w(x) \leq 1 \)
\( f(x) = (1-w(x))g(x) + w(x)h(x), \text{ the evaluation function} \)
\( \text{father}(x) = \text{the immediate predecessor } x \text{ as found by HPA} \)
\( S = \text{set of nodes already visited by HPA; called the expanded or closed nodes} \)
\( \overline{S} = \text{set of nodes that are the immediate successors of nodes in } S \text{ but are not themselves in } S; \text{ called the open or candidate nodes} \)
1. Place s in S and generate Γ(s) placing these nodes in S. For x ∈ Γ(s), father(x) = s, g(x) = c(s,x), and f(x) = (1-w(x))c(s,x) + w(x)h(x).
2. Select n ∈ S such that f(n) is a minimum. If there are ties select among the nodes that have minimum f(n), any node with maximum g(n).
3. Place n in S and for all x ∈ Γ(n) such that x is not in S compute g(x) = g(n) + c(n,x) and f(n) = (1-w(x))g(x) + w(x)h(x). If x is not in S or if it is and the new computation of f(x) is less than the old, place x in S with value f(x) and father(x) = n.
4. If n = t halt, otherwise go back to step 2.

HPA upon successfully finding the terminal node has constructed the deductive chain t, father(t), father(father(t))...father(...father(t)...)= s, with path cost g(t). HPA’s total computational effort in finding this path is proportional to the cost of building the set S. In the normal case where unit edge costs are used, search effort is proportional to |S|. This simple observation is critical to evaluating the various forms of search procedure that are proposed.

HPA is an algorithm whose ancestry combines combinatorial search procedures and artificial intelligence search procedures. When w(x) = 0 and the graphs being searched have a cardinality metric then the algorithm is similar to the Moore maze searching algorithm. When more general costs are allowed, it is a form of the Dijkstra two-point shortest path algorithm. When w(x) = 1, then the algorithm is a form of the original graph traverse of Doran and Michie. The graph traverser inaugurates the linking of combinatorial search algorithms and heuristic search algorithms. Previous artificial intelligence researchers made strenuous efforts to develop terminologies which obscured this important affinity.

The next achievements are a first attempt at a theory of search efficiency that centered around the notion of “admissibility.” A search algorithm is admissible if the solution path it finds are of minimum cost. Hart, Nilsson and Raphael (Hart, et al., 1968) devised the algorithm A* which is a form of HPA restricted to w(x) = 0.5. They restricted their attention to heuristic functions which were guaranteed to return a lower bound on the remaining cost of a solution path. A* using such a heuristic is provably guaranteed to find a least costly solution path. Two remarks are in order: (i) h(x) = 0 is trivially a lower bound and gives us back Dijkstra’s algorithm; (ii) A* is a form of the branch-and-bound principle which is a key method in enumerative combinatorial programming.

The above developments led to a natural generalization of these methods HPA (Pohl, 1969,1970a). This formulation revealed two critical questions: (i) what are the best values for w(x); (ii) how is search efficiency dependent on the accuracy of h. The first question could not be posed in terms of the predecessors to HPA. The second question received significant practical testing in the work of the graph traverser and a limited theoretical treatment within the context of A* performing admissible searches. The proper formulation of a theory of error in heuristics and its effect on search performance was a chief accomplishment of the HPA work. This issue continues to be, along with the question of incor-
porating high level planning into search procedures, the critical research ques-
tions in heuristic search.

ADMISSIBILITY AND SEARCH EFFICIENCY

To discuss search efficiency and its relation to the accuracy of the heuristic
information employed in directing search, we introduce some further termi-
nology.

\[
K(m,n) = \text{the cost of an optimal path from } m \text{ to } n \\
h^*(n) = K(n,t), \text{ to be called the } \textit{perfect} \text{ heuristic} \\
\text{If } \forall x \in X (h(x) \leq h^*(x)) \text{ then } h \text{ is an } \textit{admissible} \text{ heuristic} \\
\text{If } \exists x, y \in X (h(x) - h(y) \leq k(x, y) \text{ and } h(t) = 0) \text{ then } h \text{ is a } \textit{consistent} \text{ heuristic.}
\]

\textit{Theorem}

If HPA with \( w(x) = \frac{1}{2} \) is used with a consistent heuristic, then all
nodes included in \( S \) will have \( g(x) = k(s, x) \). (Hart, Nilsson, and
Raphael, 1968).

\textit{Corollary}

The above scheme will compute an optimal path to a terminal
node \( t \), hence it is an admissible search algorithm.

\( A^* \) is described in a way that allows nodes in set \( S \) to be re-evaluated and if
necessary placed back in \( S \). Under this additional computational burden, \( A^* \)
is admissible if its heuristic is admissible. This is not the case for HPA with
\( w(x) = \frac{1}{2} \) which for admissibility requires that the heuristic used be consistent
(exercise: show how HPA with \( w(x) = \frac{1}{2} \) and \( h \) admissible can lead to a non-
optimal solution path).

Consistency is a strong constraint and whereas \( h(x) = 0 \) trivially satisfied
consistency, it may be hard to demonstrate for more informed heuristics. An
alternate property which is more easily checked is the \textit{monotone} criterion.

A heuristic function satisfies the monotone criterion if for all \( x \in X \) and
\( y \in \Gamma(x) \), \( 0 \leq h(x) - h(y) \leq c(x, y) \) and \( h(t) = 0 \).

\textit{Theorem}

HPA with \( w(x) = \frac{1}{2} \) and \( h \) admissible and monotone will only in-
clude nodes in \( S \) for which \( g(x) = k(s, x) \).

\textit{Proof}

By induction on the number of nodes placed in set \( S \).
\( n = 1 \): The first node placed in \( S \) is \( s \) with \( g(s) = 0 \); so the theorem
is trivially true in this case.
Assume true for \( k < n \) to prove for \( n \).
Let \( x \) be the \( n \)th node placed in set \( S \) and \( g(x) > k(s, k) \) and the
PROBLEM-SOLVING AND DEDUCTION

path from s to x is $s = x_1, x_2, \ldots, x_T = x$. There is a shortest path from $s$ to $x$ namely $s = y_1, y_2, \ldots, y_p = x$. Let $y_{k}$ be the last node along this path which is in $S$. So $k < p$, otherwise $x$ would have been placed in $S$ with its optimal cost. The evaluation function for $y_{k+1}$ is

$$f(y_{k+1}) = \frac{1}{2}\left(g(y_k) + h(y_{k+1}) + c(y_k, y_{k+1})\right)$$

claim: $f(y_{k+1}) < f(x)$

First we show that $f(y_k) \leq f(y_{k+1})$.

By the monotone criterion $h(y_k) - h(y_{k+1}) \leq c(y_k, y_{k+1})$

$$f(y_{k+1}) = \frac{1}{2}\left(g(y_k) + h(y_{k+1}) + c(y_k, y_{k+1})\right)$$

$$f(y_k) = \frac{1}{2}\left(g(y_k) + h(y_k)\right)$$

$2(f(y_k) - f(y_{k+1})) = h(y_k) - (y_{k+1}) - c(y_k, y_{k+1}) \leq 0$ and so

$$f(y_k) \leq f(y_{k+1})$$ and by transitivity,

$$f(y_{k+1}) < f(y_p) < f(x_t).$$

Therefore the node $y_{k+1}$ will be included in $S$ before $x$, a contradiction, $x$ must be included in $S$ only when $g(x) = k(s, x)$.

ERROR ANALYSIS

The computational effort required by HPA is proportional to the number of times the inner loop is executed, which is equivalent to $|S|$. This set upon successful termination of HPA must be at least the size of the number of nodes on the shortest solution path. Indeed this performance is possible when HPA uses $h^*$ with $w(x) \geq \frac{1}{2}$ (Pohl, 1970a). The usual situation is less fortunate as for most problems the heuristics will be in error. We would like a precise basis for formulating the concept of error and a methodology for investigating its consequent effect on search. This theory was first elaborated in (Pohl, 1969; Pohl, 1970b).

We analyze search behavior by adopting the worst case norm for measuring the efficiency of our algorithms. Namely, given all possible inputs conforming to a problem statement what is the most effort the algorithm will require for any input in this set. This is the most widely applied norm in the complexity literature. While in many instances this norm is not as relevant as average performance, it allows a significant reduction in the difficulty of proving results.

Worst case analysis often is performed with the aid of an oracle or adversary strategy (Knuth, 1973; Kirkpatrick, 1974). Since the efficiency of an algorithm depends on how certain tests are decided, the adversary attempts to provide results which are consistent with the problem constraints but which maximally degrade performance.

We restrict our attention to heuristic functions of bounded error $\varepsilon = 0, 1, \ldots$, as applied to tree domains in search of solution paths of length $k$.  

62
\[ V \in X \{ h^*(x) - \varepsilon \leq h(x) \leq h^*(x) + \varepsilon \} , \]

\( h(x) \) is said to be of bounded error \( \varepsilon \).

*Oracle for HPA used in computing worst-case analysis*

If \( x \) is along a shortest path, \( h(x) = h^*(x) + \varepsilon \), otherwise \( h(x) = h^*(x) - \varepsilon \).

It can be proved that this oracle leads to worst-case performance for HPA acting on tree domains (Pohl, 1970b).

*Example:*

HPA will be used on a binary tree domain with \( k = 2 \), \( \varepsilon = 1 \), and \( w(x) = 1 \).

The goal node is marked by the "T" and the start node is node 1.

The internal numbering represents the order in which nodes are placed in \( S \). The external numbering represents the oracles response to evaluating \( h(x) \).

If the above analysis is extended to the general case with HPA using \( w(x) = 1 \), pure heuristic search, then all nodes \( 2 \varepsilon - 1 \) off the shortest path will be included in \( S \) (Pohl, 1970b, Theorem 8). So for binary trees, pure heuristic search in the worst case expands \( 2^{k+1} \) nodes, \( e \geq 1 \), where \( k \) is the length of the solution path. A similar analysis of HPA with \( f(x) = \frac{1}{2}(g(x)+h(x)) \) has HPA expanding \( 2^k \) nodes in the worst case.

The derivation of these two results reveals an unsuspected theoretical advantage in favor of utilizing the \( g(x) \) term in the evaluation function. Until the time of this result and empirical results backing up this relationship, all artificial intelligence researchers who confined their attention to finding non-optimal cost solutions to deductive problems believed that pure heuristic search was the most
efficient mechanism. Else, they had not given any thought to the problem.

HPA and like search algorithms can be amended to search problem spaces bi-directionally (Pohl, 1971). HPA is a uni-directional algorithm searching in the forward direction, out from s. It is often possible when t is explicitly given and $\Gamma^{-1}(x)$ the predecessor relation is available to search back from t looking for s. In symmetric graphs this is accomplished by renaming s as t and vice-versa. It is possible to combine both a forward and a backward search by i) using a choice rule to decide on which direction to search on a given iteration and ii) stopping when some node n is found in the intersection of the two searches.

The motivation for doing this is clear—search effort grows exponentially with the depth of search; two searches roughly half the depth of one longer search provides exponential savings easily offsetting the cost of the extra administrative work. This is in fact the case for the ordinary shortest path problem (Pohl, 1969) where no heuristic information is used. However, a number of experiments with bi-directional search using heuristic information have led to worse performance than uni-directional search.

It is of interest to see if error analysis of the bi-directional case can point out the difficulties. We embed the solution path in a portion of sufficiently large binary tree and perform worst case analysis. We will do an example with pure heuristic search and with the bi-directional choice rule of ‘alternating on each iteration’.

**Example:**

$$k = 3, \ e = 1, \ w(x) = 1 \ (\text{both directions})$$

- s - start node forward direction
- t - terminal node - start node backward direction
- x - nodes closed in the forward direction
- $\bar{x}$ - nodes closed in the backward direction
Under the above assumptions, bi-directional heuristic search expands a few more nodes than uni-directional search—those nodes which would ordinarily be successors of \( t \) would not be examined by uni-directional search. Note, worst case analysis with respect to functions of bounded error leads to search effort proportional to path length multiplied by a term exponentially related to error. Thus bi-directional search does not yield exponential savings as growth is not exponential in terms of path length. So heuristic search with reasonably effective search functions in the worst case does not benefit from bi-directional methods.

Let us perform our worst case analysis with respect to a different bound on heuristic error. We will say that a heuristic function \( h(x) \) has relative error \( \delta \) if

\[
V_{x \in X} \left( h^*(x)(1-\delta) \leq h(x) \leq h^*(x)(1+\delta), \quad 0 \leq \delta < 1 \right).
\]

In many ways this is more realistic than bounded error. It conforms to the common sense expectation that estimating bigger distances will lead to larger error.

Applying a worst case analysis within these bounds on the heuristic function gives results which even more strongly than in the bounded error case favor uni-directional methods. In the binary tree space of the previous example define \( T_j \) as the subtree of nodes which hang off the \( j \)th node along the solution path and which are expanded by HPA in the worst case.

\[
\text{Total number of nodes expanded} = \sum_{j=0}^{k} |T_j| + k + 1
\]

By the worst case assumptions (again may be derived from an oracle which maximally penalizes nodes along the solution path and maximally favors nodes off the solution path) the height of \( T_j \) is

\[
\text{height } T_j = \left[ \frac{2K\delta}{1 - \delta} - 2 \cdot \left( \frac{1}{1 - \delta} \right) \right].
\]

\[
|T_j| = 2^{(\text{height } T_j + 1)} - 1.
\]

So in the case of relative error, path length enters in exponentially as well as error. Bi-directional search in tree spaces under these assumptions is roughly twice as expensive as uni-directional search. This theoretical result is in surprising agreement with observed experiment (Pohl, 1971).

We introduce some further notation which is needed in discussing bi-directional search.

\[
\begin{align*}
  h_s(x) &= \text{estimator of } k(s,x) \\
  h_t(x) &= \text{estimator of } k(t,x) \\
  g_s(x) &= \text{distance found in forward search from } s \text{ to } x \\
  g_t(x) &= \text{distance found in backward search from } t \text{ to } x \\
  h(x,y) &= \text{estimator of } k(x,y)
\end{align*}
\]
PROBLEM-SOLVING AND DEDUCTION

In the just concluded argument bi-directional search uses in the forward direction \( h_1(x) \) and in the backward direction \( h_2(x) \) to guide search. Each search is being guided to the opposite end-point. One form of improvement suggested in (Pohl, 1971) was for both searches to aim at a common midpoint. Or in the absence of a convenient means of formulating a hypothesized midpoint, the evaluation function could be computed with respect to newly selected endpoints. These could be the pair \((s', t')\) such that \( g(s') + g(t') + h(s', t') \) is a minimum among all pairs of nodes \( s \in S, t \in T \). Recent work has been performed according to these suggestions (de Champeaux and Sint, 1974). The results were promising in that bi-directional methods employing these ideas on the 15-puzzle typically found shorter solution paths than the use of the unidirectional method with the same heuristics. However, the number of nodes expanded was comparable and this meant that the unidirectional method was more efficient as it does not incur the additional administrative costs that bi-directional search requires; including the cost of evaluating \( h(x, y) \) for all node pairs in the open sets.

The above remarks suggest that there may be an appropriate number of iterations in a single direction before switching to the opposite direction. This would be related to the accuracy of the heuristic function. It would seem that a more accurate heuristic function could take advantage of more frequent alternation of search direction and updating of endpoint pairs. This point is partially supported by the experiments with “look-ahead” search (Rosenberg and Kestner, 1972). Instead of computing \( T(x) \), they computed \( T^k(x) \) and only then computed the evaluation function for open nodes. They found that in a number of instances the additional effort of computing \( T^k(x) \) was offset by the improvement in choice of which next node to expand, i.e., look-ahead acts to improve the accuracy of the heuristics being used. It is worth pointing out that this technique resembles the GT-4 utilization of macro-moves (Michie, 1971; Ross, 1973).

PRACTICAL APPLICATION TO A DIFFICULT PROBLEM

The TSP has already been described and we now return to it as a case study of incorporating expertise into HPA. As it has already been proven that finding solutions to TSP which are within some known bound of optimal is NP, we are justified in using heuristic procedures.

To try and avoid confusion between the problem space graph and the TSP graph the latter will be written in italics, e.g., \( G(X, E) \) with edge costs \( c(x_i, x_j) \). A node \( x \) in the problem graph will contain as state description the two sets of edges \( A, R \) with meaning:

- \( e \in A \) means \( e \in E \) and \( e_i \) is to be included in any tour produced by a descendant of \( x \)
- \( e \in R \) means \( e \in E \) and \( e_j \) is not to be included in any tour produced by a descendant of \( x \)

66
A \cap R = \phi and furthermore there are no three edges of E in A such that all have as endpoint the same node of X.

The successor relation \( \Gamma(x) \) will either create a single new node with \( A(y) = e \cup A(x) \) a tour or the two nodes \( y \) and \( y' \) with

\[
A(y) = \{e\} \cup A(x), \quad R(y) = R(x) \\
A(y') = A(x), \quad R(y') = \{e\} \cup R(x)
\]

\( e \) is an edge of E not in \( A(x) \cup R(x) \) and is chosen by a "suitably expert" successor relation. By this we mean that the edge to be the next edge included in the partial tour A is somehow a reasonably intelligent choice. Any node which represents a tour is a terminal node; the first such node included in S terminates HPA.

The heuristic function to be used is the minimum spanning 1-tree (Held and Karp, 1971). A tree is a connected graph without cycles. A minimum spanning tree is the minimum cost tree over all trees which are subgraphs of G and include all nodes of G. A 1-tree is a tree over the node set \( \{x_2, x_3, \ldots, x_n\} \) plus two distinct edges with \( x_1 \) as their endpoint. A minimum (spanning) 1-tree is readily computed by computing a minimum spanning tree on the node set \( \{x_2, x_3, x_4, \ldots, x_n\} \) using Dijkstra's or Kruskal's algorithm (Aho, Hopcroft, and Ullman, 1974) and adding the two cheapest edges connected to \( x_1 \). Such a computation is order \( n^2 \).

A tour is a minimum 1-tree in which each node has degree two. A minimum 1-tree is a least costly element of a superset of tours, hence it is a lower bound on tours. We compute \( f(x) \) where \( x \) is associated with \( A(x) \) and \( R(x) \) in the following manner:

\[
g(x) = \sum_{e \in A(x)} c(e), \quad \text{the cost of the edges that must be included in a tour.}
\]

\[
h(x) = \text{the cost of a minimum 1-tree which includes the edges } A(x) \text{ and excludes the edges } R(x) \cdot g(x)
\]

Alternatively, if \( y \) is the father of \( x \) and \( x \) is a node created by adding \( e \) to the set \( A(y) \) then \( g(x) = g(y) + c(e) \), and if \( x \) is a node created by adding \( e \) to the set \( R(y) \) then \( g(x) = g(y) \); i.e., in the former case \( c(y, x) = c(e) \) and in the latter case \( c(y, x) = 0 \). With these definitions of \( g \) and \( h \), it is readily shown that \( h \) is monotone and therefore \( f(x) = \frac{1}{2} g(x) + \frac{1}{2} h(x) \) is admissible.

The heuristic function just defined is often not accurate, but it is possible by the following gradient technique to improve its accuracy.

Given a real n-vector \( \bar{v} = (v_1, v_2, \ldots, v_n) \) we modify the edge costs in \( G \),

\[
c_{\bar{v}}(x_i, x_j) = c(x_i, x_j) + v_i + v_j.
\]

It is obvious that if \( \pi(n) \) is a tour in \( G \) of cost \( c(\pi(n)) \) then it is a tour in \( G \) with
respect to edge costs \(c_0\) of cost \(c(\pi(n)) + 2 \sum_{i=1}^{n} v_i\). So the same tour is optimal with respect to any such modification of costs by a vector \(v\). Now a tree is not a tour in so far as there are nodes whose degree differs from 2. It is desirable to penalize with a large value of \(v_j\) a node \(x_j\) whose degree in a 1-tree exceeds two. In a like manner it is desirable to reward with a negative value of \(v_j\) a node \(x_j\) whose degree in a 1-tree is one. This can be done by computing the \(i+1\)st spanning tree from the original costs modified by \(v_i\). This tree has nodes whose degree is \(d_j^{i+1}, d_2^{i+1}, \ldots, d_n^{i+1}\). Then compute \(v_i^{i+1}\) as the sum \(v_k^{i+1} = v_k^i + d_k^{i+1} - 2\). The degrees themselves provide a reasonable means of penalizing nodes. Among the sequence of 1-trees a best estimate is provided by that tree whose cost, in relation to the original edge cost, is maximum. One terminates this iteration when either a 1-tree is a tour or when no further improvement is expected.

To complete our description of HPA using the above functions to solve TSP we must give a rule for deciding which edge will be used for generating successor nodes in the problem graph. It will be that edge, which when removed from the spanning 1-tree computation, causes its recomputation to maximally increase in cost–in effect a form of “killer heuristic.” It is also necessary to discuss the values of \(w(x)\). Using \(w(x) = \frac{1}{2}\) as has been stated gives an admissible algorithm akin to branch-and-bound. Since the \(h\) just described is a guaranteed lower bound it can serve no useful purpose to consider values \(0 \leq w(x) < \frac{1}{2}\). However \(w(x) > \frac{1}{2}\) will lead to possibly non-optimal solutions.

This over weighting of the heuristic term allows us to find a computational compromise between the two standard computational catastrophes of heuristic search procedures. That is if \(w(x) = \frac{1}{2}\) (or near \(\frac{1}{2}\)) we are guaranteed to produce a solution which is optimal (or near optimal) but because the search is too breadth-first we run out of space/time resources. If \(w(x)\) is near 1, we have pure heuristic search which in this case is likely to examine few nodes in \(G\) but is also likely to generate an unacceptably bad tour. We compromise by using a \(w(x)\) which balances these two needs.

So far in this discussion \(w(x)\) has been a fixed constant in any particular application. The above considerations lead naturally to a dynamic computation of the weighting term. In dynamic weighting in relation to admissible heuristic functions we set \(w(x) = \frac{1}{2} + \beta\) where \(x\) is a direct successor of \(s\). When we are deeper into the problem graph, further away from \(s\), we reduce \(w(x)\) to \(\frac{1}{2}\). This reduction is proportional to the distance away from \(S\). Dynamic weighting produces better solutions than uniformly overweighting the heuristic function. It is almost as efficient in terms of nodes expanded because it only becomes breadth-first as it nears a solution node. Dynamic weighting compensates for the difficulty-phenomenon in heuristic search. This is an observed degradation in the accuracy of most heuristics the further away from a terminal node they are
applied. The use of dynamic weighting has led to methods competitive in efficiency with the Lin (Lin, 1965) techniques for TSP, and providing firmer guarantees on the optimality of the resulting solutions (Pohl, 1973). Similar ideas are applicable to game tree searches such as the Harris bandwidth search (Harris, 1973).

Ramblings

The combinatorial explosion (Lighthill, 1973) is readily identified as a chief obstacle to effective problem solving. The two main tools of artificial intelligence which attempt to overcome this problem are heuristic search and the dense utilization of specific knowledge. The chess player who decides on a move typically has selected a “plan” he deems suitable for his position. This plan in the absence of tactical threat may focus his attention on a single piece or more typically an area of the board. Such is the case with knight manoeuvres or with the minority attack. Tactical play being inherently local can be analyzed by a limited exhaustive search. Similarly, the mathematician begins by selecting a method of proof which appears suitable for a particular problem, such as induction or proof by contradiction. His strategy may further involve identifying useful intermediate results—lemmas which he feels tactically capable of solving. In relation to simpler sub-problems he may use locally exhaustive methods of solution.

More generally in artificial intelligence and computer science, we seek to avert the combinatorial explosion in three ways:

1. detecting efficiently solvable sub-cases
2. utilizing efficient schemes which give approximate “non-optimal” solutions
3. utilizing heuristic search techniques to facilitate general searches

Each of these methods is “knowledgable.” An instance of (1) is the chess players use of special procedures for the elementary endgames. Often the principle of king opposition selects without search the correct move. An example of (2) is the Lin technique for the TSP problem (Lin, 1965). A tour is improved by determining whether the deletion of 3 edges and their replacement will improve the tour cost. This is continued until a tour which cannot be improved according to this standard is achieved. This is always possible to compute in polynomial time. In chess a technique in this category is to perform trades when ahead in material. Instances of (3) have been the main content of this paper.

It is interesting to note that a main contributor to this theory has recently said (Nilsson, 1974 p. 18), “The problem of efficiently searching a graph has essentially been solved and thus no longer occupies AI researchers.” What could be meant by this I am not sure. On the one hand the solution to efficiently solving the two-point path problem has long been known when heuristic information is not involved. On the other hand it is the deepest unsolved problem in
complexity theory to determine whether certain searches in graphs can be done in polynomial time. Furthermore in AI there has been a recent surge in transferring or embodying heuristic search procedures into programming languages (Bobrow and Raphael, 1973). A key difficulty to implementing search procedures implicitly is the inefficiency of any general control structure capable of administrating these searches. In that this remains a key research problem for PLANNER-like systems, it is obvious that further progress on heuristic search theory is a necessary precursor to its solution.

As was stated earlier the question of high level planning is one of the central research questions in heuristic search. It has been adequately stressed in any number of forums, and one consequence is the great popularity of planning-languages. It is mildly disconcerting that much of this work ignores or is ignorant of its natural predecessors the various heuristic search systems. The second issue which is central to heuristic search theory is a need to elaborate the results of error analysis. In their description of Merlin (Moore and Newell, 1973) and its comparison to other systems which “understand” knowledge, Moore and Newell say (ibid., p. 50) “The design of Merlin offers nothing fundamental yet to cope with error... we have let the issue of error alone.” Now it is clear from their paper that error is not just the use of inaccurate heuristics in that they additionally include other forms of representational error such as the frame problem. Nevertheless while examining a wide number of standard search methods including backtracking, they do not broach the one concrete theory of error for which results exist. The effect of these omissions in the fashionable literature is to underline the significance of these research topics in heuristic search.

I will conclude by stating three principles of heuristic search.

Principle of Utilization

Heuristics are necessary for the solution of difficult problems—
even in many finite domains. Hence the process of discovering
heuristics is the key task of problem solving.

Principle of Robustness

Heuristics are approximate uses of knowledge—the fact of them
being in error affects their use. They should be employed in schemes
which can recover from error.

Principle of Adaptability

Heuristics tend to be locally good—the nearer a solution the
better. Their most effective use is adaptive (dynamic) with respect to
their accuracy.

REFERENCES

Algorithms, Addison-Wesley, Reading, Mass.


Pohl, I. (1970a) Heuristic search viewed as path finding in a graph, Artificial Intelligence 1, 193-204.

PROBLEM-SOLVING AND DEDUCTION


Pohl, I. (1975) Practical complexity and the design of algorithms. Informatie (Nederland) 17, 205-211.


