SLAC-PUB-537
January 1969
(MISC)

SOME RESULTS ON A FORMAL THEORY OF HEURISTIC SEARCH[1]

Ira Pohl

Stanford Linear Accelerator Center
Stanford University, Stanford, California

ABSTRACT

Many Artificial Intelligence problems can be represented as a path problem over a directed graph. The search for a solution path is expedited by a heuristic function evaluating the nodes reached. This paper presents some results on the efficient use of this function, and a formal characterization of the effect of error in the heuristic function.

Key Words:

Heuristic Search, Artificial Intelligence, path finding, formal theory, graph model, problem space, error analysis, learning.

---

[1]Work supported by the U.S. Atomic Energy Commission.

(Submitted to the International Joint Conference on Artificial Intelligence, May 1969, Washington, D.C.)

Introduction

One of the important general models of artificial intelligence is the directed graph[2] model [3] - [9]. In this model a node contains a description of a possible problem state. If it is possible to get from some state x to state y in a single move (rule of inference, operator, etc.) then there is a directed edge from x to y. More generally, we wish to know if a path exists between two nodes. We distinguish one as the initial node and look for a path to the other, designated the goal node. Such a path is called the solution to our problem. Sometimes we are interested in the shortest path between two nodes, but normally any solution path will be acceptable.

The work of Amarel [3][4], Michie and Doran [5][6][7], and Hart, Nilsson, and Raphael [8][9] contributed to different aspects of formulating problems in this model. Amarel worked principally on the representation of different problems in this model. Michie and Doran have developed a general problem-solving program, called the Graph Traverser, for finding paths using heuristic functions to control the search for the goal node. Hart, Nilsson and Raphael have given sufficient conditions on heuristic functions to guarantee that a class of path finding algorithms will find the shortest solution path in the space. Each of these researchers have contributed important results and continue to do so. The above brief treatment of their work is of course insufficient, but is intended to serve as a context for this paper's further elaboration and formalization of this model.

[2] For graph theory terminology, see Berge [1] or Ore [2].

Problem Spaces and Heuristic Search

A directed graph G is a set of nodes X and a mapping Γ from the nodes into themselves.

$$G: \quad X = \{x_1, x_2, \ldots, x_n\}$$

$$\Gamma: \quad X \rightarrow X$$

$$E = \{(x_i, x_j) \mid x_i \in X \wedge x_j \in \Gamma(x_i)\}$$

The size or cardinality of the graph is denoted by $|G|$ and can be unbounded. When using directed graphs to characterize problem domains we attach to each x_i a vector $\vec{v}_i$ which contains the complete description of the problem. For example, in the case of the 15 puzzle (see Fig. 1) a vector describing it would be (9, 5, 1, 3, 13, 7, 2, 8, 14, 6, 4, 11, 10, 15, 12, 0) where 0 denoted the blank position. The mapping Γ would represent possible single moves from one problem state to another. In this domain we are at some initial node (or set of nodes) and wish to reach some goal node (or goal set). We must produce a path from the initial node to the goal node. Purely exhaustive methods are impractical in complicated spaces with $|G|$ and $|E|$ large or possibly infinite. In most instances we have heuristics which aid in narrowing the search. For our discussion heuristic information is a function over the state vectors $\vec{v}_i$ into the non-negative reals.

An Algorithm for Heuristic Search

When solving most artificial intelligence problems we are not ordinarily interested in the most 'elegant' or shortest path,[3] but in how to obtain any path cheaply. A search method visits a number of nodes in G to find a path. We want this number to be as few as

[3] See [8] and [10] for results in regard to this restriction.

possible, so that it may be computationally feasible to find solution paths which are inherently long; i.e., the shortest path is long.

HPA - Heuristic Path Algorithm

s = initial node

t = goal node

$g(x)$ = the number of edges from s to x, as found in our search

$h(x)$ = an estimate of the number of edges from x to t, our heuristic function

$f(x) = g(x) + \omega \cdot h(x)$ $\qquad 0 \leq \omega \leq \infty$

By convention if $\omega = \infty$ then $f(x) = h(x)$

S = the nodes that have been visited

$\tilde{S}$ = the nodes which can be reached from S along an edge, but are not in S

1. Place s in S and calculate $\Gamma(s)$, placing them in $\tilde{S}$.

   If $x \in \Gamma(s)$ then $g(x) = 1$ and $f(x) = 1 + \omega \cdot h(x)$.

2. Select $n \in \tilde{S}$ such that f(n) is a minimum.

3. Place n in S and $\Gamma(n)$ in $\tilde{S}$ (if not already in $\tilde{S}$) and calculate f for the successors of n.

   If $x \in \Gamma(n)$ then $g(x) = 1 + g(n)$ and

   $f(x) = g(x) + \omega \cdot h(x)$.

4. If n is the goal state then halt, otherwise go to step 2.

Note: HPA builds a tree; as each node is reached a pointer to its predecessor is maintained. Upon termination the solution path is traced back from the goal node through each predecessor.

HPA is a typical path finding algorithm and is similar to the algorithms used in the work of Michie and Doran, and Hart, Nilsson and Raphael. It will find a path if one exists and the graph is finite (a proof of this is similar to Theorem 1 in [8]), and can fail if the graph is infinite. In the Graph Traverser [6] only h is used, by our convention $\omega = \infty$. The intuitive reason for this weighting is that prior distance in reaching a node is so much "water over the dam". Indeed, if h is an accurate estimator of distance from the goal, it will indicate the node nearest the goal. This remaining distance is what determines the fewest nodes to visit. This argument is plausible, but relies on the accuracy of the heuristic function. Any space for which an accurate estimator exists is a solved problem domain. Only domains with inaccurate estimators are interesting, and it is these cases for which the efficient use of heuristic information is necessary. In Table 1 we list some common weights and the type of search produced.

Theorem 1. If h is perfect (exact, correct) then for $\omega \geq 1$, the search by HPA is optimal, i.e., visits the fewest nodes possible.

Pf.

Case 1. $\omega = \infty$.

Let the shortest path be k steps long. $\mu = (s, x_1, \ldots, x_k)$, $x_k = t$

Since h is perfect then $h(x_i) = k - i$. Now when s is expanded $x_1 \in \Gamma(s)$ and since other nodes are off the shortest path they must have an h value greater than $k - 1$. So x_1 is picked on the next iteration, and is expanded in turn. At each iteration the node along the shortest path and currently in $\tilde{S}$ is placed in S. Therefore only nodes on the shortest path are expanded, and so our method is optimal.

Case 2. $\omega = 1$.

The argument is the same as above, except that along μ, $f(x_i) = g(x_i) + h(x_i)$ $= i + k - i = k$. So along the shortest path, all nodes evaluate to k, and other nodes evaluate to greater than k.

Case 3. $1 < \omega < \infty$.

Consider $f^*(n) = \frac{f(n)}{\omega} = \frac{g(n)}{\omega} + h(n)$. Each step from s adds $1/\omega$ from the first term, and along the shortest path the second term is reduced by 1. Now $1/\omega < 1$, so along shortest path f* decreases with $f^*(x_j) = k - j + j/\omega$. Each node along μ decreases in value by $1 - 1/\omega$ while nodes off μ increase by at least $1/\omega$. Thus f* will expand only the nodes on μ, and so is optimal. Now f* determines the same search order as f, so f is optimal.

We see that for h perfect and for $\omega \geq 1$ HPA only expands nodes on the shortest path. If $\omega < 1$ then other nodes may be expanded, with $\omega = 0$ the worst case. This case is given in Table 1 and produces an exhaustive parallel search. It now remains to investigate cases where h is in error.

Heuristic Error

To do this rigorously will require easily analyzable spaces. However, as will become clear, this should shed much light on the use of heuristic search, where previously only heated "intuitively" justifiable arguments were used. The spaces used will be regular infinite rooted trees with unique goal nodes. The root is the only node without predecessors, and a regular tree is one in which each node has the same number of descendants.

The simplest such space is the unary tree (Fig. 2a). Over this space all functions, representing any heuristic function and weighting, are equivalent. The search always proceeds from node 1 to node 2, ... until the goal node is encountered. This case is without interest and we move on to the binary tree space (Fig. 2b). This is already non-trivial and complex enough to represent reasonable problem domains such as Lisp programs [11].

Theorem 1 applies regardless of the specific directed graph structure, thus the use of a perfect h in our evaluator f is optimal for $1 \leq \omega < \infty$. Perhaps no

heuristic information exists for our domain, and we therefore have h identically zero throughout the binary tree space. The evaluators we could use are then:

(a) $f = g + \omega \cdot h = g$ $\quad (h = 0)\ 0 < \omega < \infty$

(b) $f = h = 0$ $\quad \omega = \infty$

The use of g constitutes a parallel search, while the use of 0 is a search where all the nodes in $\tilde{S}$ (open nodes) will be tied. At each iteration, step 2 of HPA will randomly choose from the nodes tied, and therefore (b) produces a random search. If instead our tie-break rule was first-in/first-out (FIFO) we would have parallel search. Last-in/first-out (LIFO) would be a depth first rule.

Theorem 2. Over an infinite binary tree, a parallel search on average requires $2^k + 2^{k-1} - 1/2$ nodes expanded to find a node k steps from the root.

Pf.

The number of nodes in a binary tree of diameter k (maximum length from the root) is $B_k = 2^{k+1} - 1$. Since a node may be anywhere along the kth level with equal probability, we must search $B_k + 1$ to B_{k+1} nodes with the average being

$$\frac{1}{2}(B_k + 1 + B_{k+1}) = \frac{1}{2}(2^k + 2^{k+1} - 1) = 2^k + 2^{k-1} - \frac{1}{2}$$

Lemma: A binary tree of diameter k has $2^{k+1} - 1$ nodes.

Pf. By induction.

Case k = 0: B_0 is just the root node. $B_0 = 2^{0+1} - 1 = 2 - 1 = 1$.

Case k = 1: B_1 is just the root node and two successors. $B_1 = 3 = 2^{1+1} - 1$

Inductive step: Assume $B_k = 2^{k+1} - 1$, to show $B_{k+1} = 2^{k+2} - 1$.

Each level has 2^k nodes, where k is the distance from the root.

$$B_{k+1} = B_k + (k+1 \text{ level})$$
$$= B_k + 2^{k+1} = 2^{k+1} + 2^{k+1} - 1 = 2^{k+2} - 1 \quad .$$

So in a parallel search of a binary tree, we have the above formula determining, or average, how many nodes must be visited. It is exponentially varying with the distance from the root; typical behavior in complex problem spaces.

In contrast, let us examine the expected number of nodes visited by a random search in finding a goal node k steps from the root. In the simplest non-trivial case k equals 1 ($k = 0$ is trivial).

Theorem 3. Over an infinite binary tree, a random search expects to visit an unbounded number of nodes to find a goal node 1 step from the root.

Note: HPA is not told that the node is only 1 away and consequently does not restrict its search to this level.

Pf.

E = Expected number of nodes visited

r_i = Probability of finding the goal node in exactly i steps

p_i = Probability of finding the goal node on the ith step, having reached this step

ℓ_i = Probability of not finding the goal node on any step before the ith step

$$E = 1 + \sum_{i=1}^{\infty} i \cdot r_i \,, \qquad \text{The 1 is for the root node}$$

$$r_i = p_i \cdot \ell_i$$

We show

$$p_i = \frac{1}{i+1} \,.$$

With each step of HPA over a binary tree one node is removed from $\tilde{S}$ and placed in S, but two nodes are placed in $\tilde{S}$. This means at step i there are i + 1 nodes in $\tilde{S}$. In the case of f = 0 and random tie-breaking, they all are equally

likely to be picked. Furthermore since the goal node is 1 away from the root and it is always in set $\tilde{S}$ until found by HPA so $p_i = \frac{1}{i+1}$.

$$\ell_i = \frac{1}{i} , \qquad \text{we show this by induction}$$

$$\ell_i = 1 - \sum_{j=1}^{i-1} r_j , \quad \ell_1 = 1$$

$$\ell_2 = 1 - r_1 = 1 - p_1\ell_1 = 1 - \frac{1}{2} \cdot 1 = \frac{1}{2} \quad .$$

Assume $\ell_k = 1/k$, we must show $\ell_{k+1} = 1/k+1$

$$\ell_{k+1} = 1 - \sum_{j=1}^{k} r_j = 1 - \sum_{j=1}^{k-1} r_j - r_k$$

$$= \ell_k - r_k = \ell_k(1-p_k) = \frac{1}{k}\left(1 - \frac{1}{k+1}\right) = \frac{1}{k+1} \quad .$$

Therefore

$$E = 1 + \sum_{i=1}^{\infty} \left(i \cdot \frac{1}{i+1} \cdot \frac{1}{i} \right) = 1 + \sum_{i=2}^{\infty} \frac{1}{i}$$

$$= \sum_{i=1}^{\infty} \frac{1}{i} \qquad \text{the harmonic series which does not converge.}$$

This result is similar to gambler's ruin problems [12] ,[13]. Essentially the space grows too fast, and when not lucky enough to initially find the goal node, we soon find it disappearing in the growth of $\tilde{S}$.

Normally, a search is restricted by time or space limitations. This is akin to limiting our infinite space to some maximum depth. If we are interested in finding a goal node in a binary tree of diameter k, then a maximum of $2^{k+1} - 1$ nodes need be searched. If each of these nodes are with equal probability the goal node, then any exhaustive non-repeating search would yield the same expected value for

nodes visited $\frac{1}{2}(B_k + B_0)$, or 2^k. Each method would get better performance for different groups of nodes. The parallel search visits the closer nodes soonest, and for goal nodes near the root this method has a better expected value than random or depth first (LIFO) search.

In our theorem, HPA was unaware that the goal was at level 1, and so with finite probability searched portions of the space which were beyond the solution. A modification on this would be to tell our procedure that the goal was on level k. When this is known, the depth first or backtrack track method [14], [15] is optimal. The algorithm should go down to a depth of k and check to see if this is the goal node. If not it backs up one level and goes down to the next node at level k. It continues backing up and going forward to the next node on level k until it finds the goal. Since this search pattern looks at nodes on the kth level as soon as possible, it must be best in the sense of the expected number of nodes visited. It would be the worst search pattern if the goal node was actually at level 1. Here it either finds the goal on the first try (like any other method) or must look at half the tree before returning to the goal node. A parallel search is a conservative strategy, you are guaranteed not to penetrate below the part of the tree containing the goal, while you pay by always investigating the whole subtree up to that level.

How Error Affects Heuristic Search

In general we have neither a complete lack of information nor perfectly accurate information, but instead we have a heuristic function which has error. We wish to resolve for this more typical instance, how best to use a heuristic function. To investigate this question, we stay in our binary tree space using HPA. We will do a worst case analysis in the spirit of error analysis in numerical problems.

Consider

h = perfect estimator

ϵ = a bound on the error $\quad 0, 1, 2, 3, \ldots$

h^* = actual heuristic function

$$h - \epsilon \leq h^* \leq h + \epsilon$$

We will choose values of h^* conforming to the above limits, but in such a manner as to mislead HPA to the greatest extent. In doing this, we assume that HPA will always choose the worst nodes in case of ties, i.e., nodes off the solution path. An example of this analysis is Fig. 4, where HPA just uses the h^* function as the evaluator. The order of search is according to the numbers inside the nodes with x, the goal being reached in 5 steps. To make h^* as bad as possible ($\epsilon = 1$), we add ϵ to each node on the shortest path, and we subtract ϵ from each node off the shortest path. If h itself was used HPA would only visit the 3 nodes on the shortest path which is a consequence of Theorem 1.

One of the principal questions is the comparison between h^* and $g + h^*$ as evaluators. Both to get more of a flavor of our error analysis and some inklings as to this comparison, we work through the example in Fig. 5. Let us examine HPA using $f = h^* + g$, as in Fig. 5b. At the goal node x,

$$h(x) = 0, \qquad g(x) = 1$$

$$h^*(x) = h(x) + \epsilon = 0 + 2 = 2$$

$$f(x) = 3 \ ;$$

while at node 2 we have

$$h(2) = 2, \qquad g(2) = 1$$

$$h^*(2) = h(2) - \epsilon = 2 - 2 = 0$$

$$f(2) = 1 \ .$$

Node 3 has

$$h(3) = 3, \qquad g(3) = 2$$

and so both have increased by 1 from the values of its predecessor node 2.

Thus

$$h^*(3) = 3 - 2 = 1$$

$$f(3) = 3$$

an increase of 2 from its predecessor. In contrast when using only h^* (Fig. 5a), f increases by 1. This allows the search in Fig. 5b to cut-off sooner along an incorrect path. The results of Fig. 5 are:

distance to goal = 1

maximum error $\epsilon = 2$

nodes visited $f = h^*$ are 9

nodes visited $f = h^* + g$ are 5

We can generalize this result and find a formula giving the number of nodes visited for different errors and path lengths in our binary tree space.

Theorem 4. Let k be the distance from the root node to the goal node and $f = g + h^*$ be the function used by HPA, then the maximum number of nodes visited in our binary tree space is

$$2^{\epsilon} \cdot k + 1 \quad .$$

Pf.

If the goal node is distance k from the root, then if h^* is perfect HPA visits $k + 1$ nodes (Theorem 1). In the worst case with an error of ϵ, all nodes ϵ off the shortest path will be visited, excluding the nodes succeeding the goal node. This is shown in our discussion of Fig. 5.

Case $\epsilon = 0$.

This is as stated above $k + 1$.

Case $\epsilon = 1$.

These are the nodes on the shortest path plus those one off the shortest path. There are k nodes one off the shortest path so we have

$$k + k + 1 = 2k + 1 \quad .$$

Case $\epsilon \geq 2$.

At this point each unexpanded node (leaf) has two not yet explored successors. The number of leaves in a binary tree grows as $2^{\epsilon-1} \cdot k$. So the tree for error $\epsilon \geq 2$ is

$$\underbrace{2k+1}_{\epsilon=1} + 2 \cdot k + 2^2 k + \ldots + 2^{\epsilon-1} k$$

$$= 1 + 2k + k \sum_{i=1}^{\epsilon-1} 2^i$$

$$= 1 + 2^{\epsilon} \cdot k$$

Similarly we prove

Theorem 5. Let k be the distance from the root node to the goal node and $f = h^*$ be the function used by HPA, then the maximum number of nodes visited in our binary tree space is

$$\begin{cases} \frac{4^{\epsilon}}{2} \cdot k + 1 & \epsilon \geq 1 \\ k + 1 & \epsilon = 0 \end{cases}$$

Pf.

Case $\epsilon = 0$.

Again by Theorem 1 a path to a goal node distance k from the root is found by HPA visiting $k + 1$ nodes, if h^* is perfect.

Case $\epsilon = 1$.

Here as in the previous theorem HPA visits nodes 1 off the solution path and we have

$$2k + 1 = \frac{4^1}{2} k + 1 \qquad \text{nodes visited.}$$

Case $\epsilon \geq 2$.

After the first level each increment in the error allows a maximum of two additional levels to be visited. This is because along the solution path we use $h + \epsilon$ and off the solution path we use $h - \epsilon$ giving a 2ϵ leeway. The trees with the maximum number of explored nodes are

$$\underbrace{2k + 1}_{\epsilon=1} + \underbrace{2k + 2^2k}_{\epsilon=2} + \underbrace{2^3k + 2^4k}_{\epsilon=3} + \ldots$$

$$+ \underbrace{2^{2\epsilon-3}k + 2^{2\epsilon-2}k}_{\epsilon}$$

$$= 1 + 2k + k \sum_{i=1}^{2\epsilon-2} 2^i = 1 + \frac{4^\epsilon}{2} \cdot k$$

These results suggest two plausible conclusions for general heuristic search.

1. The more accurate h* is the fewer nodes HPA visits.
2. It is better to include g in the evaluator.

Furthermore the results are extendable to any tree structured problem space with a unique goal node of interest. Namely

Theorem 6. If HPA is searching any tree structured space for some goal node then

(a) $f = h^*$ will visit at least as many nodes $f = g + h^*$ in the sense of the above worst case analysis.

(b) If $h - \epsilon_1 \leq h_1^* \leq h + \epsilon_1$ and $h - \epsilon_2 \leq h_2^* \leq h + \epsilon_2$, $\epsilon_2 > \epsilon_1$. Then the number of nodes visited by HPA using h_2^* will be at least as many as

when using h_1^* in the sense of the worst case and with ω being the same for both evaluators.

Pf.

Part (b) is obvious.

Part (a) follows from Theorems 4 and 5.

The major defect of the above analysis and consequently the generality of the results is that problem spaces are ordinarily not trees. In a tree each node has a unique path back to the root. Problem domains have circuits normally and many alternate solution paths. Also the above analysis is for the worst case and while these results are attainable, in practice they are unlikely. It is important to monitor the behavior of HPA in actual problems.

Empirical Treatment

Each problem space and each heuristic function for this space presents a problem in selecting an appropriate ω. Just using g guarantees finding the shortest solution path, however, a price is paid in the breadth of the search. On the other hand, just using h may be unstable; HPA then runs down the search tree to great depths before changing to another part of the space (behavior analogous to Theorem 3). It is appropriate to search for a middle ground, which our results from Theorems 4 and 5 indicate should be near a $\omega = 1$.

Let us consider a specific heuristic function, h*, used by HPA to solve problems in some problem space. We characterize its effectiveness by the number of nodes N it visits in finding a solution path of length k. We may summarize this information in calculating its branch rate.[4] The branch rate ρ is the number of successors a node has in a regular tree of diameter k and cardinality N. So

[4]This is a suggestion of N. Nilsson.

given a particular heuristic function and a weighted evaluator we solve a number of sample problems in our space obtaining for each solution

ρ_ω = branch rate for this weighting

K_ω = solution path length

where

$1 \leq \rho_\omega \leq$ average degree of the problem domain.

We then use this information to appropriately select ω for our evaluator. It is even possible to use simple learning to obtain this value. From my own experiments on the 15 puzzle [10], the number of nodes visited varies significantly with ω.

Conclusion

Until now the analysis of heuristic search has been mostly "heuristic." The complexity of the imprecise notion has impeded rigorous analysis. Hopefully, this paper has shown this need not be. A start has been made on a mathematical theory of heuristic search. The most interesting results have been to indicate the usefulness of including the g function for efficient heuristic search and a mathematical treatment based on worst case error analysis.

References

1. C. Berge, The Theory of Graphs and its Applications, (Metheun Co., Ltd., London, 1962).
2. O. Ore, Theory of Graphs, (AMS Colloquium Publications, Providence, R. I., 1962); Vol. 38.
3. S. Amarel, "On machine representations of problems of reasoning about actions," Carnegie Institute of Technology, Pittsburgh, Pa. (June 1966).
4. S. Amarel, "An approach to heuristic problem solving and theorem proving in the propositional calculus," Carnegie Institute of Technology, Pittsburgh, Pa. (June 1966).
5. J. Doran and D. Michie, "Experiments with the Graph Traverser program," Proceedings of the Royal Society (A), 294, (1966); pp. 235-259.
6. J. Doran, "An approach to automatic problem-solving," Machine Intelligence 1, (Oliver and Boyd, Edinburgh, 1967); pp. 105-123.
7. D. Michie, "Strategy building with the Graph Traverser," Machine Intelligence 1, (Oliver and Boyd, Edinburgh, 1967); pp. 105-123.
8. P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," IEEE Trans. on System Sciences and Cybernetics (July 1968).
9. N. Nilsson, "Searching problem-solving and game-playing trees for minimal cost solutions," IFIP Congress (1968); (preprints) pp. H125-H130.
10. I. Pohl, "Computationally efficient search over discrete spaces," Ph.D. Dissertation, Stanford University, Stanford, California (expected June 1969).
11. J. McCarthy, LISP 1.5 Programmer's Manual, (MIT Press, Cambridge, Mass. 1964).
12. I. Pohl, "Phrase-structure productions in PL/I," CACM 10, (1967); p. 757.

13. W. Feller, An Introduction to Probability Theory and its Applications, (John Wiley and Sons, Inc., New York, 1950).

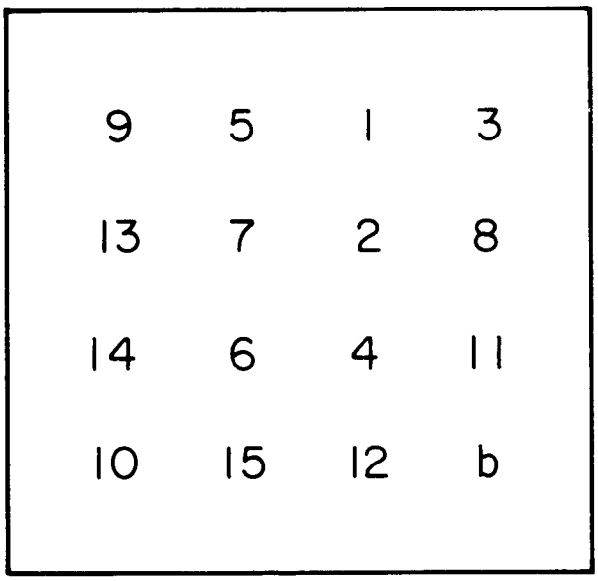
14. S. Golomb and L. Baumert, "Backtrack programming," JACM 12, (October 1965); pp. 516-524.

15. R. Floyd, "Nondeterministic algorithms," JACM 14, (October 1967); pp. 636-644.

TABLE 1

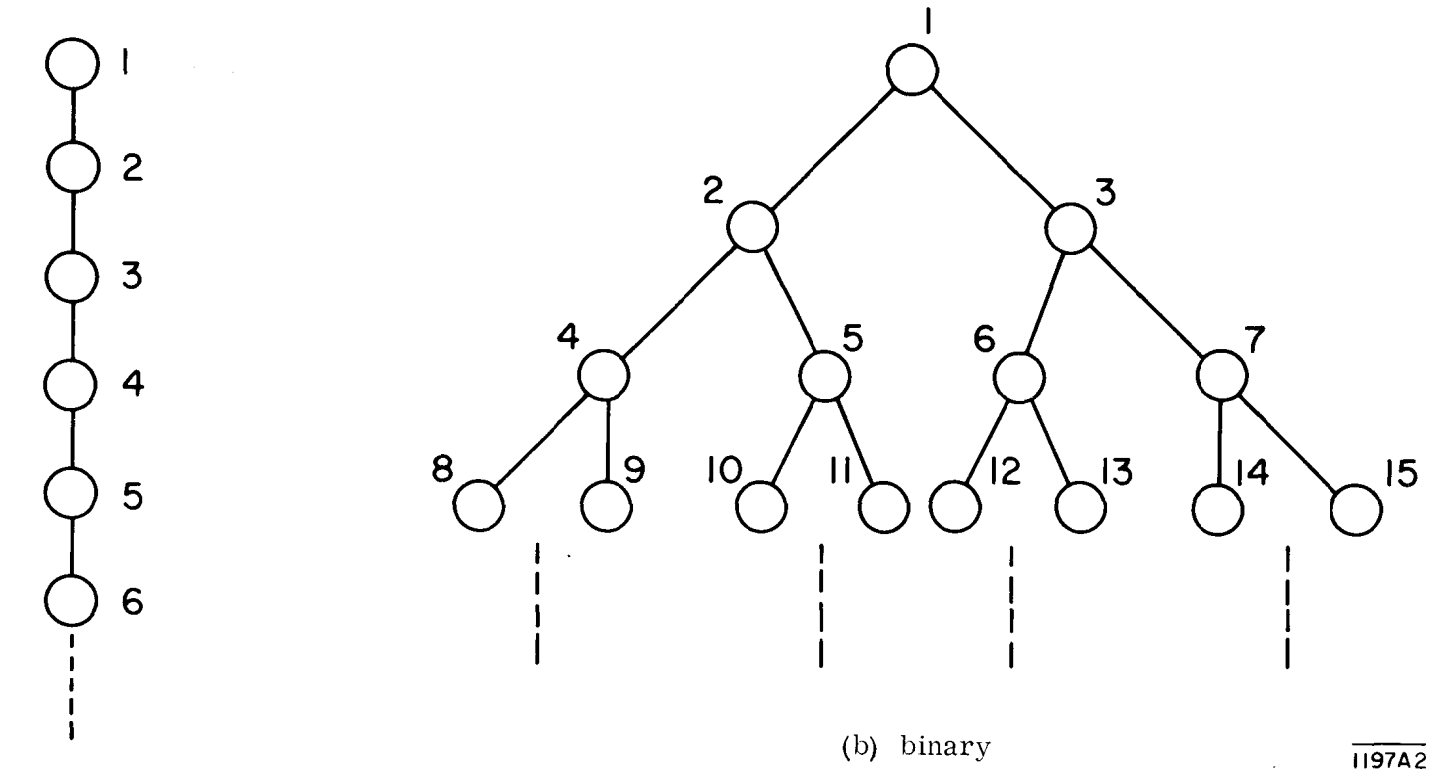
COMMONLY USED EVALUATORS

$\omega = 0$,	$f(n) = g(n)$	exhaustive parallel or breadth first search
$\omega = \infty$,	$f(n) = h(n)$	simple or pure heuristic search - Graph Traverser
$\omega = 1$,	$f(n) = g(n) + h(n)$	compound heuristic search



1197A1

FIG. 1--15 puzzle



(a) unary

(b) binary

FIG. 2--Regular infinite trees.

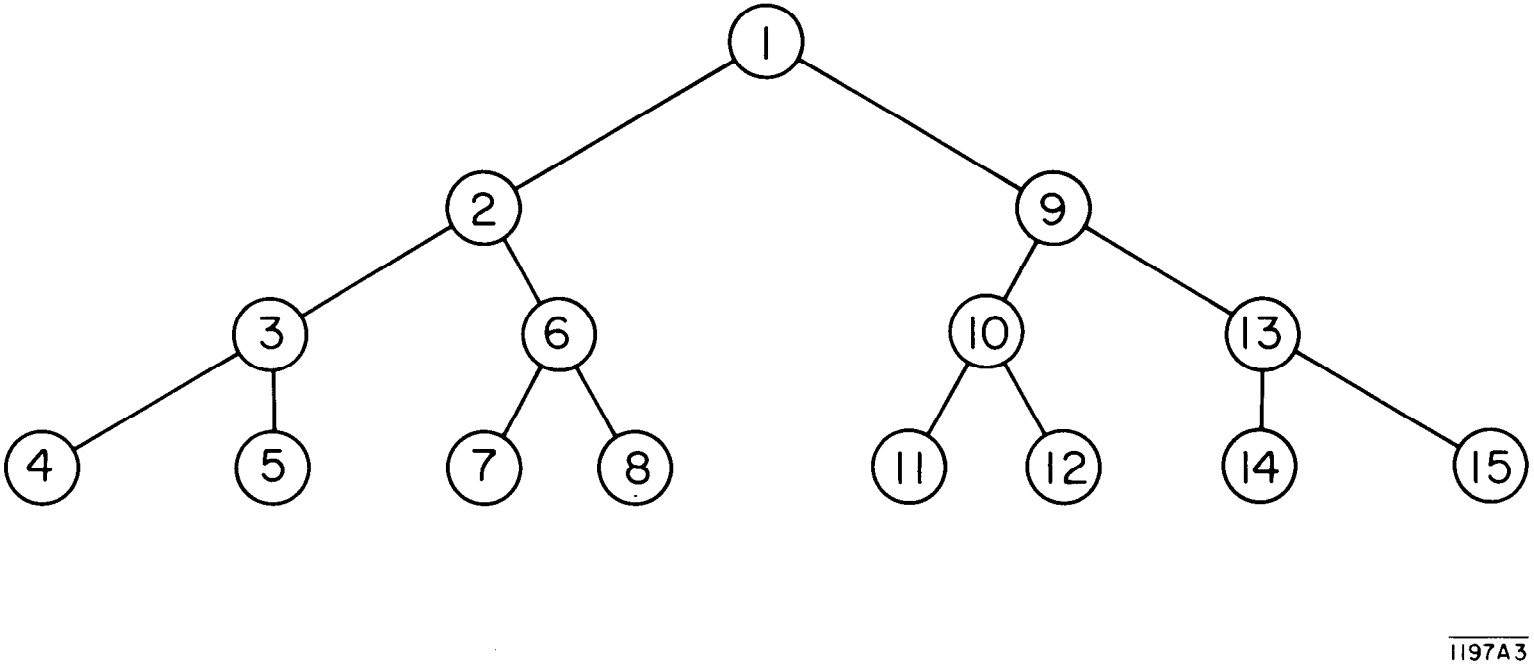


FIG. 3--Nodes numbered in order visited by a depth first search to level 3.

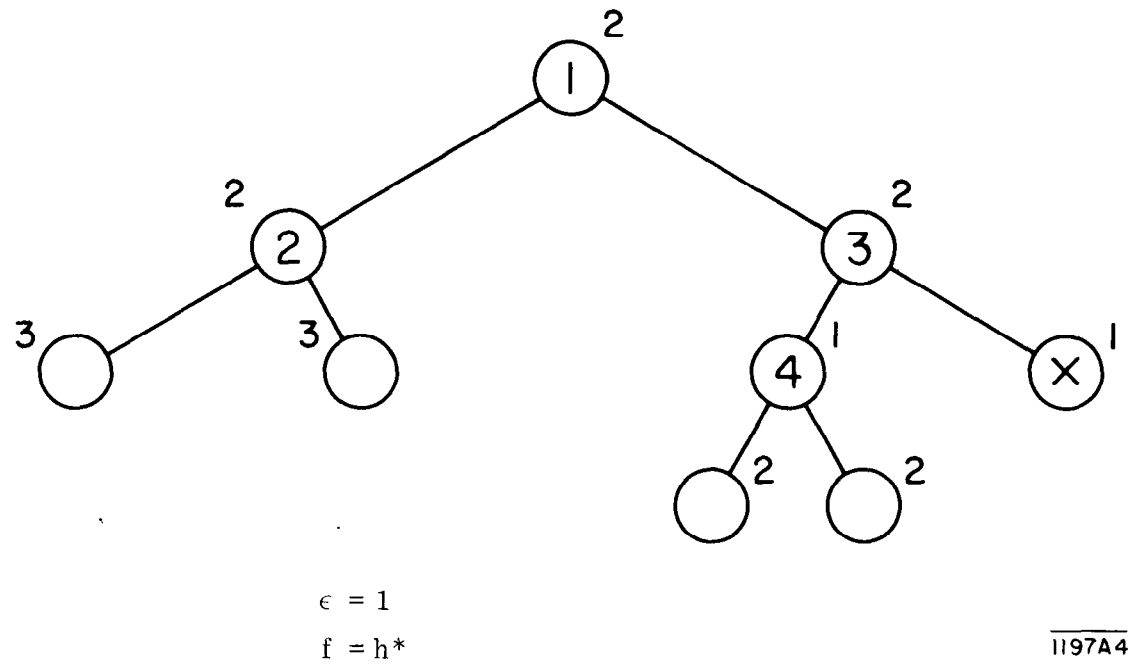
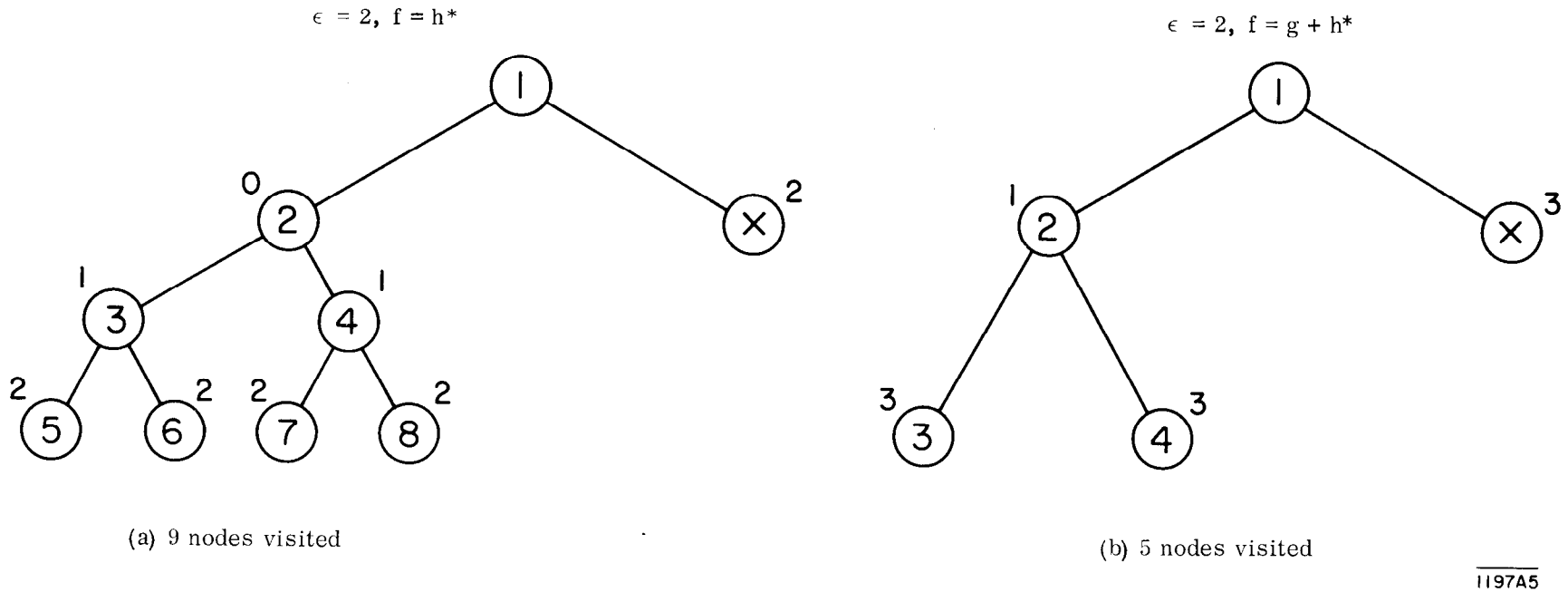


FIG. 4--The goal node is marked by an x. Other nodes are labeled by order of search (inside) and f value outside. Five nodes are searched when x is found.



(a) 9 nodes visited

(b) 5 nodes visited

1197A5

FIG. 5--Comparison between two possible ωt's for h*:
(a) $\omega t = \infty$ (b) $\omega t = 1$