the "look-ahead" effort in searching game trees. Procedures developed via the heuristic approach generally have not been able to guarantee that minimum cost solution paths will always be found.

This paper draws together the above two approaches by describing how information from a problem domain can be incorporated in a formal mathematical approach to a graph analysis problem. It also presents a general algorithm which prescribes how to use such information to find a minimum cost path through a graph. Finally, it proves, under mild assumptions, that this algorithm is optimal in the sense that it examines the smallest number of nodes necessary to guarantee a minimum cost solution.

The following is a typical illustration of the sort of problem to which our results are applicable. Imagine a set of cities with roads connecting certain pairs of them. Suppose we desire a technique for discovering a sequence of cities on the shortest route from a specified start to a specified goal city. Our algorithm prescribes how to use special knowledge—e.g., the knowledge that the shortest road route between any pair of cities cannot be less than the airline distance between them—in order to reduce the total number of cities that need to be considered.

First, we must make some preliminary statements and definitions about graphs and search algorithms.

B. Some Definitions About Graphs

A *graph* G is defined to be a set $\{n_i\}$ of elements called nodes and a set $\{e_{ij}\}$ of directed line segments called arcs. If e_{pq} is an element of the set $\{e_{ij}\}$, then we say that there is an *arc* from node n_p to node n_q and that n_q is a *successor* of n_p. We shall be concerned here with graphs whose arcs have *costs* associated with them. We shall represent the cost of arc e_{ij} by c_{ij}. (An arc from n_i to n_j does not imply the existence of an arc from n_j to n_i. If both arcs exist, in general $c_{ij} \neq c_{ji}$.) We shall consider only those graphs G for which there exists $\delta > 0$ such that the cost of every arc of G is greater than or equal to δ. Such graphs shall be called δ *graphs*.

In many problems of interest the graph is not specified explicitly as a set of nodes and arcs, but rather is specified implicitly by means of a set of source nodes $S \subset \{n_i\}$ and a successor operator Γ, defined on $\{n_i\}$, whose value for each n_i is a set of pairs $\{(n_j, c_{ij})\}$. In other words, applying Γ to node n_i yields all the successors n_j of n_i and the costs c_{ij} associated with the arcs from n_i to the various n_j. Application of Γ to the source nodes, to their successors, and so forth as long as new nodes can be generated results in an explicit specification of the graph thus defined. We shall assume throughout this paper that a graph G is always given in implicit form.

The subgraph G_n from any node n in $\{n_i\}$ is the graph defined implicitly by the single source node n and some Γ defined on $\{n_i\}$. We shall say that each node in G_n is *accessible* from n.

A *path* from n_1 to n_k is an ordered set of nodes $(n_1, n_2, \ldots, n_k)$ with each n_{i+1} a successor of n_i. There exists a path from n_i to n_j if and only if n_j is accessible from n_i. Every path has a cost which is obtained by adding the individual costs of each arc, $c_{i,i+1}$, in the path. An *optimal path* from n_i to n_j is a path having the smallest cost over the set of all paths from n_i to n_j. We shall represent this cost by $h(n_i, n_j)$.

This paper will be concerned with the subgraph G_s from some single specified *start node* s. We define a nonempty set T of nodes in G_s as the *goal nodes*.[1] For any node n in G_s, an element $t \in T$ is a *preferred* goal node of n if and only if the cost of an optimal path from n to t does not exceed the cost of any other path from n to any member of T. For simplicity, we shall represent the unique cost of an optimal path from n to a preferred goal node of n by the symbol $h(n)$; i.e., $h(n) = \min_{t \in T} h(n,t)$.

C. Algorithms for Finding Minimum Cost Paths

We are interested in algorithms that search G_s to find an optimal path from s to a preferred goal node of s. What we mean by searching a graph and finding an optimal path is made clear by describing in general how such algorithms proceed. Starting with the node s, they generate some part of the subgraph G_s by repetitive application of the successor operator Γ. During the course of the algorithm, if Γ is applied to a node, we say that the algorithm has *expanded* that node.

We can keep track of the minimum cost path from s to each node encountered as follows. Each time a node is expanded, we store with each successor node n both the cost of getting to n by the lowest cost path found thus far, and a pointer to the predecessor of n along that path. Eventually the algorithm terminates at some goal node t, and no more nodes are expanded. We can then reconstruct a minimum cost path from s to t known at the time of termination simply by chaining back from t to s through the pointers.

We call an algorithm *admissible* if it is guaranteed to find an optimal path from s to a preferred goal node of s for any δ graph. Various admissible algorithms may differ both in the order in which they expand the nodes of G_s and in the number of nodes expanded. In the next section, we shall propose a way of ordering node expansion and show that the resulting algorithm is admissible. Then, in a following section, we shall show, under a mild assumption, that this algorithm uses information from the problem represented by the graph in an optimal way. That is, it expands the smallest number of nodes necessary to guarantee finding an optimal path.

II. An Admissible Searching Algorithm

A. Description of the Algorithm

In order to expand the fewest possible nodes in searching for an optimal path, a search algorithm must constantly make as informed a decision as possible about which node to expand next. If it expands nodes which obviously cannot be on an optimal path, it is wasting effort. On the other hand, if it continues to ignore nodes that might be on an

[1] We exclude the trivial case of $s \in T$.

optimal path, it will sometimes fail to find such a path and thus not be admissible. An efficient algorithm obviously needs some way to evaluate available nodes to determine which one should be expanded next. Suppose some *evaluation function* $\hat{f}(n)$ could be calculated for any node n. We shall suggest a specific function below, but first we shall describe how a search algorithm would use such a function.

Let our evaluation function $\hat{f}(n)$ be defined in such a way that the available node having the smallest value of $\hat{f}$ is the node that should be expanded next. Then we can define a search algorithm as follows.

Search Algorithm A:*

1) Mark s "open" and calculate $\hat{f}(s)$.
2) Select the open node n whose value of $\hat{f}$ is smallest. Resolve ties arbitrarily, but always in favor of any node $n \in T$.
3) If $n \in T$, mark n "closed" and terminate the algorithm.
4) Otherwise, mark n closed and apply the successor operator Γ to n. Calculate $\hat{f}$ for each successor of n and mark as open each successor not already marked closed. Remark as open any closed node n_i which is a successor of n and for which $\hat{f}(n_i)$ is smaller now than it was when n_i was marked closed. Go to Step 2.

We shall next show that for a suitable choice of the evaluation function $\hat{f}$, the algorithm A^* is guaranteed to find an optimal path to a preferred goal node of s and thus is admissible.

B. The Evaluation Function

For any subgraph G_s and any goal set T, let $f(n)$ be the actual cost of an optimal path *constrained to go through* n, from s to a preferred goal node of n.

Note that $f(s) = h(s)$ is the cost of an unconstrained optimal path from s to a preferred goal node of s. In fact, $f(n) = f(s)$ for every node n on an optimal path, and $f(n) > f(s)$ for every node n not on an optimal path. Thus, although $f(n)$ is not known a priori (in fact, determination of the true value of $f(n)$ may be the main problem of interest), it seems reasonable to use an estimate of $f(n)$ as the evaluation function $\hat{f}(n)$. In the remainder of this paper, we shall exhibit some properties of the search algorithm A^* when the cost $f(n)$ of an optimal path through node n is estimated by an appropriate evaluation function $\hat{f}(n)$.

We can write $f(n)$ as the sum of two parts:

$$f(n) = g(n) + h(n) \tag{1}$$

where $g(n)$ is the actual cost of an optimal path from s to n, and $h(n)$ is the actual cost of an optimal path from n to a preferred goal node of n.

Now, if we had estimates of g and h, we could add them to form an estimate of f. Let $\hat{g}(n)$ be an estimate of $g(n)$. An obvious choice for $\hat{g}(n)$ is the cost of the path from s to n having the smallest cost so far found by the algorithm. Notice that this implies $\hat{g}(n) \geq g(n)$.

A simple example will illustrate that this estimate is easy to calculate as the algorithm proceeds. Consider the

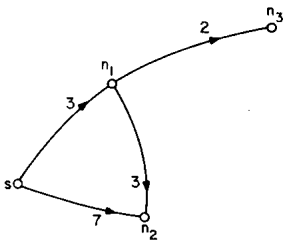

Fig. 1.

subgraph shown in Fig. 1. It consists of a start node s and three other nodes, n_1, n_2, and n_3. The arcs are shown with arrowheads and costs. Let us trace how algorithm A^* proceeded in generating this subgraph. Starting with s, we obtain successors n_1 and n_2. The estimates $\hat{g}(n_1)$ and $\hat{g}(n_2)$ are then 3 and 7, respectively. Suppose A^* expands n_1 next with successors n_2 and n_3. At this stage $\hat{g}(n_3) = 3 + 2 = 5$, and $\hat{g}(n_2)$ is lowered (because a less costly path to it has been found) to $3 + 3 = 6$. The value of $\hat{g}(n_1)$ remains equal to 3.

Next we must have an estimate $\hat{h}(n)$ of $h(n)$. Here we rely on information from the problem domain. Many problems that can be represented as a problem of finding a minimum cost path through a graph contain some "physical" information that can be used to form the estimate $\hat{h}$. In our example of cities connected by roads, $\hat{h}(n)$ might be the airline distance between city n and the goal city. This distance is the shortest possible length of any road connecting city n with the goal city; thus it is a lower bound on $h(n)$. We shall have more to say later about using information from the problem domain to form an estimate $\hat{h}$, but first we can prove that if $\hat{h}$ is any lower bound of h, then the algorithm A^* is admissible.

C. The Admissibility of A*

We shall take as our evaluation function to be used in A^*

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n) \tag{2}$$

where $\hat{g}(n)$ is the cost of the path from s to n with minimum cost so far found by A^*, and $\hat{h}(n)$ is any estimate of the cost of an optimal path from n to a preferred goal node of n. We first prove a lemma.

Lemma 1

For any nonclosed node n and for any optimal path P from s to n, there exists an open node n' on P with $\hat{g}(n') = g(n')$.

Proof: Let $P = (s = n_0, n_1, n_2, \cdots, n_k = n)$. If s is open (that is, A^* has not completed even one iteration), let $n' = s$, and the lemma is trivially true since $\hat{g}(s) = g(s) = 0$. Suppose s is closed. Let Δ be the set of all closed nodes n_i in P for which $\hat{g}(n_i) = g(n_i)$. Δ is not empty, since by assumption $s \in \Delta$. Let n^* be the element of Δ with highest index. Clearly, $n^* \neq n$, as n is nonclosed. Let n' be the successor of n^* on P. (Possibly $n' = n$.) Now $\hat{g}(n') \leq \hat{g}(n^*) + c_{n^*,n'}$ by definition of $\hat{g}$; $\hat{g}(n^*) = g(n^*)$ because n^* is in Δ, and $g(n') = g(n^*) + c_{n^*,n'}$ because P is an optimal path. Therefore, $g(n') \leq g(n')$. But in