On the Complexity of Admissible Search Algorithms

Alberto Martelli

Istituto di Elaborazione della Informazione del Consiglio Nazionale delle Ricerche, Via S. Maria 46, 56100 Pisa, Italy

Recommended by Nils Nilsson and Ira Pohl

ABSTRACT

This paper analyzes the complexity of heuristic search algorithms, i.e. algorithms which find the shortest path in a graph by using an estimate to guide the search. In particular, algorithm A, due to Hart, Nilsson and Raphael, is shown to require $O(2^N)$ steps, in the worst case, for searching a graph with N nodes, if the so called "consistency assumption" does not hold for the estimate. Furthermore, a new search algorithm is presented which runs in $O(N^2)$ steps in the worst case and which never requires more steps than A*.*

1. Introduction

Heuristic search is one of the core topics of artificial intelligence (Nilsson [10, 11]). Its importance is based on the concept of "state-space representation", according to which different problems can be transformed into the canonical problem of finding a path through a space of problem states from the initial state to a goal state. In order to solve this search problem in an efficient way, several search techniques have been developed which use heuristic information, i.e. special knowledge available from the problem domain.

Among the first examples of heuristic search algorithms we have the Graph Traverser (Michie and Ross [9]), and algorithm A* by Hart et al. [2], which gives a good formalization of the main concepts of heuristic search. Generalizations of these algorithms have been described by Pohl [13] and Harris [1].

Very similar ideas have been developed in other fields such as operations research. For instance, Martelli and Montanari [8] have shown that each dynamic programming problem can be transformed into a shortest path problem by suitably defining the cost of a path. Furthermore, the branch-and-bound method (Lawler and Wood [5]) is a shortest path method which uses heuristic estimate to guide the search.

In this paper we analyze the complexity of heuristic search algorithms by computing the number of steps which are required for searching a graph of a

Artificial Intelligence 8 (1977), 1–13

certain size. Pohl [12, 13] has considered other aspects of the complexity of heuristic search algorithms, by computing the number of steps with respect to the error of the heuristic function, when the search is performed on a tree.

In order to make our analysis more precise, we consider in this paper only "admissible" search algorithms, i.e. algorithms which find a minimal-cost path. In particular we show that, in the worst case, algorithm A* can run for $O(2^N)$ steps when the search is performed on a graph with N nodes and the so called "consistency assumption" does not hold. Furthermore, we give a new search algorithm and we show that, in the worst case, it runs for $O(N^2)$ steps and that it never requires more steps than A*.

2. Algorithm A*

Let G be a directed graph with a *start node* s, a set of *goal nodes* and a positive cost $c(n_i, n_j)$ associated with every arc (n_i,n_j). We want to find a minimal-cost path between the start node and a goal node, where the cost of a path is given by the sum of the costs of its arcs.

Algorithm A*, due to Hart et al. [2, 3], is a propagation algorithm, i.e. it starts with node s and generates step by step the graph G until a goal node is reached. In order to speed up the search of the optimal path, the propagation is guided by *heuristic information*, i.e. information available from the problem represented by the graph The heuristic information is taken into account by associating a *total estimate* $\hat{f}(n)$ with each node n. At each step, the node n whose total estimate $\hat{f}(n)$ is smallest is selected and its successors are generated (this process is called *expansion* of the node).

The total estimate $\hat{f}(n)$ is an estimate of the cost of a minimal cost path from s to a goal node constrained to go through node n, and can be expressed as

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

where $\hat{g}(n)$ is an estimate of the cost $g(n)$ of a minimal cost path from s to n and $\hat{h}(n)$ is an estimate of the cost $h(n)$ of a minimal cost path from n to a goal node. Here, $\hat{g}(n)$ is constructed step by step by the algorithm, whereas $\hat{h}(n)$ is obtained from the heuristic information.

A slightly modified version of algorithm A* is given by the following sequence of steps.

(1) Put the start node s on a list called OPEN. Set

$$\hat{g}(s) \leftarrow 0 \quad \text{and} \quad \hat{f}(s) \leftarrow \hat{h}(s).$$

(2) If OPEN is empty exit with failure; otherwise continue.

(3) Remove from OPEN that node n whose $\hat{f}$ value is smallest and put it on a list called CLOSED. (Resolve ties for minimal $\hat{f}$ values arbitrarily, but always in favor of any goal node.)

(4) If n is a goal node, exit with the solution path obtained by tracing back through the pointers; otherwise continue.

(5) Expand node n, generating all of its successors. (If there are no successors, go to (2).) For each successor n_i, compute

$$g_i \leftarrow \hat{g}(n) + c(n, n_i).$$

(6) If a successor n_i is not already on either OPEN or CLOSED, set

$$\hat{g}(n_i) \leftarrow g_i \quad \text{and} \quad \hat{f}(n_i) \leftarrow g_i + \hat{h}(n_i).$$

Put n_i on OPEN and direct a pointer from it back to n.

(7) If a successor n_i is already on OPEN or CLOSED and if $\hat{g}(n_i) > g_i$, then update it by setting

$$\hat{g}(n_i) \leftarrow g_i \quad \text{and} \quad \hat{f}(n_i) \leftarrow g_i + \hat{h}(n_i).$$

Put n_i on OPEN if it was on CLOSED and redirect to n the pointer from n_i.

(8) Go to (2).

It is possible to prove (Nilsson [10]) that if, for every node n, $\hat{h}(n)$ is a lower bound on the cost $h(n)$ of a minimal cost path from node n to a goal node, then algorithm A* is *admissible*, i.e. it always finds an optimal path. Moreover, it is possible to simplify algorithm A* by making a further assumption on the estimate $\hat{h}$: for any two nodes m and n which are connected by an arc (m, n) we have

$$\hat{h}(m) - \hat{h}(n) \leqslant c(m, n).$$

This assumption is called the *consistency assumption* and its meaning is that, by moving from a node to any successor, we must always have a better estimate.

When the consistency assumption holds, it is possible to prove (Nilsson [10]) that, when A* expands a node n, it has already found an optimal path to n. Therefore, algorithm A* will never reopen a closed node and thus it can be implemented in a simpler way.

A further property which can be proved (Hart et al. [3]) for algorithm A* is the following: every node n which is expanded by A* must also be expanded by any other admissible algorithm A such that A* is more informed than A. We say that A* is more informed than A when the estimate $\hat{h}(n)$ used by algorithm A* is strictly larger (for each nongoal node n) than the estimate used by algorithm A. According to this result, algorithm A* is optimal with respect to the number of distinct expanded nodes; note however that the same node can be expanded several times during the search and thus the total number of expansions can be larger for algorithm A* than for algorithm A.

The above criterion of optimality can be accepted in some artificial intelligence applications when the process of expanding a node for the first time is expensive with respect to the cost of subsequent expansions. However in this paper we assume that all expansions of a node have comparable costs and thus we take as criterion of optimality the total number of expansions performed by the algorithm, i.e. the number of iterations of the algorithm. In this case algorithm A* is no longer optimal and in Section 4 we shall give a new algorithm which is better than A*. Algorithm A* is optimal according to both criteria only when the consistency assumption holds, since no node can be expanded more than once.

3. The Complexity of A*

In this section we make a worst case analysis of algorithm A*, i.e. we give an upper bound to the running time of this algorithm depending on the size of the graphs to which it is applied. In order to do this, we assume as a basic operation the expansion of a node, and thus we characterize algorithm A* with the number of expansions it takes to search some graph of size N. Note that the size of a graph cannot be simply the number of nodes of the graph. In fact, algorithm A* expands only a subset of the nodes and it can work even when the graph is infinite. Therefore we give the following definitions.

Let a *search graph* be a directed graph $G = (V,A)$, finite or infinite, with a start node s, a set of goal nodes, a cost attached to every arc and such that an estimate $\hat{h}(n)$ is defined for each node $n \in V$. Let $\overline{V} \subseteq V$ be a subset of the nodes defined as follows:

(1) The start node s belongs to $\overline{V}$.

(2) A node n belongs to $\overline{V}$ if $g(n)+\hat{h}(n) \leqslant h(s)$ (where $h(s)$ is the cost of a minimal cost path) and if at least one parent node belongs to $\overline{V}$.

It is easy to verify that, if the estimate $\hat{h}$ is a lower bound on h, then algorithm A* will expand only nodes belonging to $\overline{V}$ and, therefore, a good choice for the size N of the graph is the number of nodes in $\overline{V}$.

Let us analyze now the complexity of algorithm A* when the consistency assumption holds. Given a graph of size $N = |\overline{V}|$, A* will expand at most all the nodes of $\overline{V}$. (In fact A* will possibly not expand some nodes for which $g(n)+\hat{h}(n) = h(s)$.) Moreover, as we pointed out before, when a node is expanded, it will never be reopened and thus every node of $\overline{V}$ is expanded at most once. Therefore algorithm A* with the consistency assumption expands fewer than N nodes and, hence, it runs in $O(N)$ steps.

Although the consistency assumption is a quite reasonable assumption, in some cases it is not satisfied by the estimate. For instance, Martelli and Montanari [6] give two different estimates for the same problem, such that a nonconsistent estimate is better than a consistent one. Furthermore, in many practical applications, the condition on the estimate $\hat{h}$ to be a lower bound on h is abandoned in favor of better estimates, which allow a more efficient search. Usually, in these cases, the estimate does not satisfy the consistency assumption as well. An example of such estimates is given by Harris [1] for the traveling salesman problem.

If we abandon the consistency assumption, then every node of $\overline{V}$ can be reopened several times and the running time of A* can become much greater than N. In fact, in this section we will show that the running time of A* can become exponential. This result is stated in the following theorem.

THEOREM 3.1. *For all N there exists a search graph G_N of size N, with positive costs and estimates which are lower bounds ($\hat{h}(n) \leqslant h(n)$ for each n), on which algorithm* A* *runs for $O(2^N)$ steps.*

The proof of this theorem is based on two results by Johnson [4] and Martelli and Montanari [7]. Johnson analyzed the complexity of Dijkstra's algorithm, i.e. the algorithm which finds the minimal cost path between a given node s and all other nodes in a graph whose arcs have positive or negative costs. Dijkstra's algorithm behaves exactly like algorithm A* except that it uses no estimate ($\hat{h}(n) = 0$ for each n) and that it stops only when the list OPEN is empty. Johnson proved the following theorem.

THEOREM 3.2. *For all N there exists an N-node search graph D_N without estimate, with some arcs of negative cost but no cycle of negative cost, on which Dijkstra's algorithm runs for $O(2^N)$ steps.*

The graphs D_N for which Theorem 3.2 holds have N nodes $(n_1, \ldots, n_N)$ and all the arcs (n_i, n_j) such that $i > j$. The costs of these arcs are:

$$c_D(n_2, n_1) = -2,$$
$$c_D(n_{i+1}, n_1) = c_D(n_i, n_1) - (2^{i-2}+1), \quad 2 \leqslant i < N,$$
$$c_D(n_i, n_{j+1}) = c_D(n_i, n_j) + 1, \quad 1 < j+1 < i \leqslant N.$$

Furthermore the start node s is n_N.

An example of such graphs, for $N = 5$, is given in Fig. 1 (disregard the nodes n_0 and n_6).

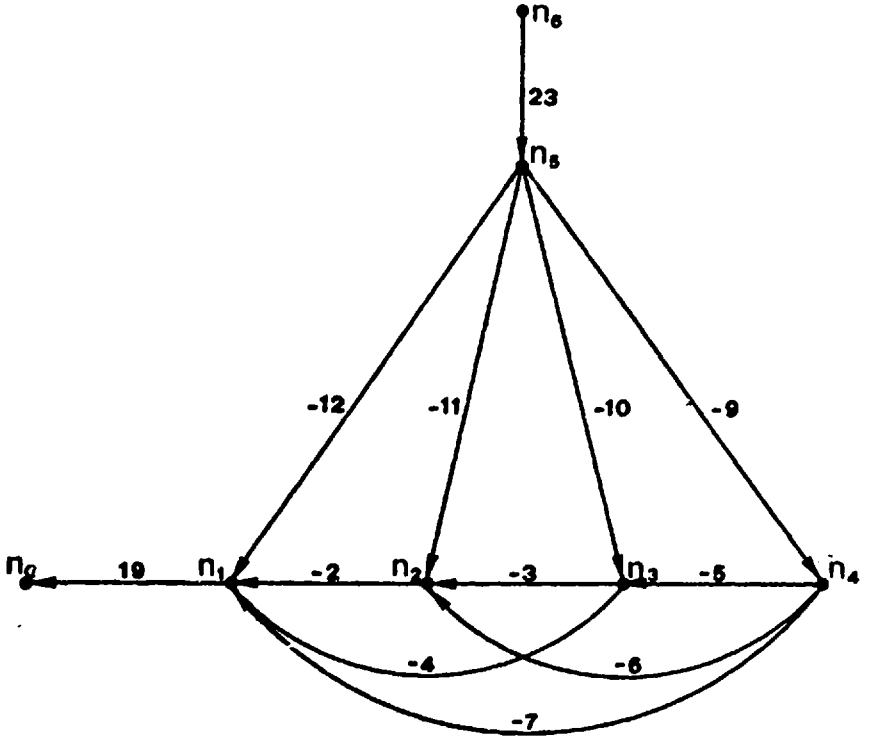

FIG. 1. An example of a graph on which Dijkstra's algorithm runs for $O(2^N)$ steps.

Upon termination, the minimal distance $g(n_i)$ between $s = n_N$ and the node n_i is:

$$g(n_N) = 0,$$
$$g(n_{i-1}) = g(n_i) - (2^{i-2}+1), \qquad N \geqslant i > 1.$$

It is also possible to verify that the cost of each arc (n_i, n_{i-1}) is

$$c_D(n_i, n_{i-1}) = -(2^{i-2}+1), \qquad 1 < i \leqslant N,$$

and thus

$$g(n_i) = \sum_{k=i}^{N-1} c_D(n_{k+1}, n_k), \qquad 1 \leqslant i < N.$$

Furthermore the minimal cost path between the node n_i and the node n_{i-k} is $n_i, n_{i-1}, \ldots, n_{i-k+1}, n_{i-k}$.

Let us now define a search graph D'_N as a slight modification of D_N obtained by adding two nodes, n_0 and n_{N+1}, and two arcs, (n_1, n_0) and (n_{N+1}, n_N) with the following costs

$$c_D(n_1, n_0) = -\sum_{i=1}^{N-1} c_D(n_{i+1}, n_i),$$

$$c_D(n_{N+1}, n_N) = -\sum_{i=1}^{N-1} c_D(n_{i+1}, n_i) + (N-1).$$

The graph D'_5 is given in Fig. 1.

By assuming as start node of this new graph D'_N the node n_{N+1}, we see that Dijkstra's algorithm behaves on D'_N exactly as on D_N, except that it requires two more steps. In fact, the node n_{N+1} can be closed only at the first step and the node n_0 can be closed only at the last step, since the $\hat{g}$ value of n_0 is always greater than the $\hat{g}$ value of all other nodes.

The behaviour of Dijkstra's algorithm on graph D'_5 of Fig. 1 is summarized in Table 1. In this table every column gives the $\hat{g}$ value for every node and every row corresponds to an iteration of the algorithm. The numbers enclosed by a rectangle correspond to closed nodes, whereas the underlined number in each row denotes the node which is expanded at that step.

From this table we can notice the exponential running time of Dijkstra's algorithm. In fact, node n_1 is expanded $2^3 = 8$ times, node n_2 $2^2 = 4$ times, node n_3 $2^1 = 2$ times, and nodes n_4, n_5, n_6 and n_0 once.

Notice that, if we take n_0 as the goal node and if we assign to each node n_i an estimate $\hat{h}(n_i) = 0$, then algorithm A* will run on the graphs D'_N exactly as Dijkstra's algorithm, since n_0 is the last node to be closed by the latter algorithm. Therefore, Table 1 gives also the behaviour of algorithm A*, if the numbers are now considered as the $\hat{f}$ value of each node.

The second result upon which the proof of Theorem 3.1 is based, is due to Martelli and Montanari [7] and is stated in the following theorem.

THEOREM 3.3. *Given a search graph G with positive costs and estimates, it is possible to construct a search graph D, without estimates and possibly with negative costs, such that algorithm* A* *runs on D exactly as it runs on G.*

The graph D can be obtained with the following construction: Nodes and arcs of D correspond to nodes and arcs of G, and the cost $c_D(m,n)$ of an arc (m,n) of D is given by

$$c_D(m,n) = \hat{h}_G(r)+c_G(m,n)-\hat{h}_G(m).$$

Furthermore a new node s' is added to D connected to s with an arc of cost $c_D(s',s) = \hat{h}_G(s)$ and s' becomes the start node of D.

We can now construct a family of graphs G_N which correspond to the graphs D'_N according to the above construction. Each graph G_N has $N+1$ nodes $(n_0, n_1, \ldots, n_N)$ and it has the same arcs as the graph D'_N. The estimate of each node is

$$\hat{h}_G(n_0) = \hat{h}_G(n_1) = 0,$$
$$\hat{h}_G(n_i) = \hat{h}_G(n_{i-1})+(2^{i-2}+2), \qquad 1 < i \leqslant N.$$

The cost of each arc is

$$c_G(n_i,n_j) = c_D(n_i,n_j)+\hat{h}_G(n_i)-\hat{h}_G(n_j).$$

TABLE 1.

n_6	n_5	n_4	n_3	n_2	n_1	n_0
0	-	-	-	-	-	-
[0]	23	-	-	-	-	-
[0]	[23]	14	13	12	11	-
[0]	[23]	14	13	12	[11]	30
[0]	[23]	14	13	[12]	10	30
[0]	[23]	14	13	[12]	[10]	29
[0]	[23]	14	[13]	10	9	29
[0]	[23]	14	[13]	10	[9]	28
[0]	[23]	14	[13]	[10]	8	28
[0]	[23]	14	[13]	[10]	[8]	27
[0]	[23]	[14]	9	8	7	27
[0]	[23]	[14]	9	8	[7]	26
[0]	[23]	[14]	9	[8]	6	26
[0]	[23]	[14]	9	[8]	[6]	25
[0]	[23]	[14]	[9]	6	5	25
[0]	[23]	[14]	[9]	6	[5]	24
[0]	[23]	[14]	[9]	[6]	4	24
[0]	[23]	[14]	[9]	[6]	[4]	23
[0]	[23]	[14]	[9]	[6]	[4]	[23]

It is easy to verify that $c_D(n_{N+1},n_N) = \hat{h}_G(n_N)$ and thus graphs G_N and D'_N satisfy the construction of Theorem 3.3. Therefore, according to this theorem, algorithm A* runs on G_N exactly as it runs on D'_N and thus it has exponential running time.

For instance, graph G_5 is shown in Fig. 2, where encircled numbers give the estimate $\hat{h}$, and we can verify that it corresponds to graph D'_5 of Fig. 1 according to the above construction. Furthermore we see that Table 1 gives the behaviour of algorithm A* on G_5 (by disregarding of course the column corresponding to n_6) exactly as it gives the behaviour of A* on D'_5.

By the definition of G_N it is possible to prove that the costs of the arcs are all positive and that the estimates are a lower bound on the minimal cost, thus completing the proof of Theorem 3.1.

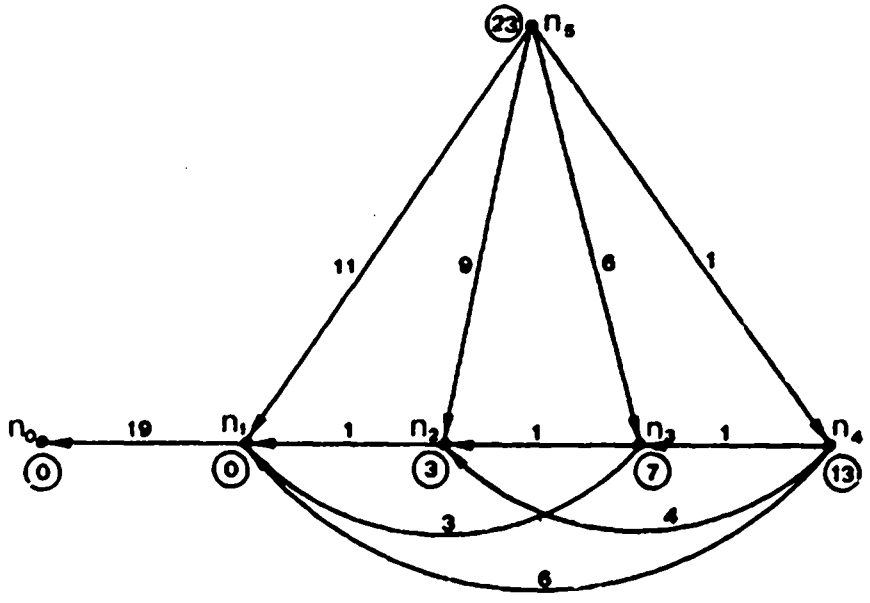

FIG. 2. An example of a graph on which algorithm A* runs for $O(2^N)$ steps.

4. A Better Search Algorithm

As pointed out before, algorithm A* has been proved optimal when the estimate is consistent. However we have shown in the previous section that, when the consistency assumption is abandoned, A* can have an exponential running time and its optimality cannot be proved any longer. Therefore we can try to modify algorithm A* in order to improve its behaviour with nonconsistent estimate. In this section we give a new search algorithm which can run for at most $O(N^2)$ steps and we show that this new algorithm never takes more steps than A*.

In defining a new algorithm for searching a graph we want, of course, an admissible algorithm, i.e. an algorithm which finds an optimal path when the estimate $\hat{h}$ is a lower bound on h. Furthermore we want the following two conditions to be always satisfied. First, we want the new algorithm to make use of the available heuristic information for reducing the search in the graph. This requirement can be

formalized by imposing that our algorithm expands only nodes belonging to the set $\bar{V}$, defined in Section 3, when $\hat{h} \leqslant h$. Note that only in this case the complexity of a new algorithm can be compared with the complexity of algorithm A*, because the two algorithms, applied to the same search graph, will expand the same subset of nodes. Second, we want that our algorithm behaves exactly as algorithm A* on search graphs for which the consistency assumption holds, since, in this case, A* is optimal.

Because of the first requirement we cannot search a graph using shortest path algorithms which are known to be polynomial but which do not use any estimate. For example, instead of using algorithm A* for searching a given graph G, we might search G with Dijkstra's algorithm, i.e. by expanding at each step the node whose $\hat{g}$ value is smallest without using the estimate. Since the costs of the arcs of G are positive, we know that Dijkstra's algorithm runs in $O(N')$ steps, where N' is the cardinality of a subset $\bar{V}'$ of the nodes of G, containing all the nodes n such that $g(n) \leqslant h(s)$. But $N' \geqslant N$ and we cannot compare the complexity of the two algorithms, since we cannot compare 2^N with N'.

Our new search algorithm, let us call it algorithm B, tries to use the estimate $\hat{h}$ only when it is really necessary to reduce the search, and otherwise it does not use it since it knows that sometimes the estimate can be nonconsistent thus giving wrong information. In order to do this, algorithm B makes use of a global variable F which, at any step, gives the greatest $\hat{f}$ value which has been achieved up to now by a closed node. The next node n to be expanded is chosen as follows: If all open nodes have an $\hat{f}$ value not smaller than F, then n is chosen as in algorithm A*. Otherwise, if there are some nodes with $\hat{f} < F$, n is chosen in the set of nodes with $\hat{f} < F$ without considering the estimate $\hat{h}$. We can do this because we know that the cost of the solution will be not smaller than F (see Lemma 3.3 in Nilsson's book [10, p. 64]) and thus any open node with $\hat{f} < F$ will be expanded sooner or later.

Algorithm B is thus a simple variant of A* and can be obtained from it by substituting steps (1) and (3) with the following steps:

(1′) Put the start node s on a list called OPEN. Set

$$\hat{g}(s) \leftarrow 0, \quad \hat{f}(s) \leftarrow \hat{h}(s), \quad F \leftarrow 0.$$

(3′) If there are some nodes in OPEN with $\hat{f} < F$, select among them the node n whose $\hat{g}$ value is smallest; otherwise, select the node n in OPEN whose $\hat{f}$ value is smallest and set $F \leftarrow \hat{f}(n)$. (Resolve ties arbitrarily, but always in favor of any goal node.) Remove n from OPEN and put it on a list called CLOSED.

The proof that this algorithm is admissible will not be given, since it would be very similar to the proof of the admissibility of A* given by Nilsson [2]. It would also be very easy to show that algorithm B fulfils the first condition stated at the beginning of this section, i.e. that it expands only nodes in $\bar{V}$. Furthermore, the second requirement (i.e. that B behaves as A* when the estimate is consistent) is also met, since, when the estimate is consistent, it is not possible to have at any step a node with $\hat{f} < F$.

In order to see how algorithm B works, we summarize in Table 2 its computation on the graph of Fig. 2. In this table the last column gives the F value at each step, and the $\hat{f}$ value of each node is represented as $\hat{g}+\hat{h}$. We see that in this case closed nodes are never reopened and thus the algorithm has linear running time. However, algorithm B can have worse behaviours, as Table 3 shows. This table displays the

TABLE 2.

n_5	n_4	n_3	n_2	n_1	n_0	F
0+23	-	-	-	-	-	0
[0+23]	1+13	6+?	9+3	11+0	-	23
[0+23]	[1+13]	2+7	5+3	7+0	-	23
[0+23]	[1+13]	[2+7]	3+3	5+0	-	23
[0+23]	[1+13]	[2+7]	[3+3]	4+0	-	23
[0+23]	[1+13]	[2+7]	[3+3]	[4+0]	23+0	23
[0+23]	[1+13]	[2+7]	[3+3]	[4+0]	[23+0]	23

TABLE 3.

n_5	n_4	n_3	n_2	n_1	n_0	F
0+0	-	-	-	-	-	0
[0+0]	1+13	6+7	9+3	11+0	-	0
[0+0]	1+13	6+7	9+3	[11+0]	30+0	11
[0+0]	1+13	6+7	[9+3]	10+0	30+0	12
[0+0]	1+13	6+7	[9+3]	[10+0]	29+0	12
[0+0]	1+13	[6+7]	7+3	9+0	29+0	13
[0+0]	1+13	[6+7]	[7+3]	8+0	29+0	13
[0+0]	1+13	[6+7]	[7+3]	[8+0]	27+0	13
[0+0]	[1+13]	2+7	5+3	7+0	27+0	14
[0+0]	[1+13]	[2+7]	3+3	5+0	27+0	14
[0+0]	[1+13]	[2+7]	[3+3]	4+0	27+0	14
[0+0]	[1+13]	[2+7]	[3+3]	[4+0]	23+0	14
[0+0]	[1+13]	[2+7]	[3+3]	[4+0]	[23+0]	23

computation of algorithm B on the same graph of Fig. 2, but where the estimate of the start node is $\hat{h}(n_5) = 0$. We see that node n_1 is closed four times, node n_2 three times, node n_3 twice and nodes n_4, n_5, n_0 once. Therefore, in this case, we have an $O(N^2)$ running time. The next theorem shows that algorithm B cannot have a worse behaviour.

THEOREM 4.1. *Given any search graph G of size N, with positive costs and estimates which are lower bounds on the minimal cost* ($\hat{h}(n) \leqslant h(n)$ *for each n), then algorithm* B *runs on it for at most* $O(N^2)$ *steps.*

Proof. Let $S = f_1, f_2, \ldots, f_r$ be the sequence of the $\hat{f}$ values of the nodes in the order in which they are closed by algorithm B, i.e. f_i gives the $\hat{f}$ value of the node closed at the ith iteration of algorithm B. Of course we have $f_1 = \hat{h}(s)$ and $f_r = h(s)$, i.e. the cost of the minimal path.

Let us take now a subsequence $\bar{S} = f_{i_1} f_{i_2} \ldots f_{i_p}$ of S constructed as follows: $i_1 = 1$ and $f_{i_{k+1}}$ is the first element of S following f_{i_k} such that $f_{i_{k+1}} \geqslant f_{i_k}$. Therefore $\bar{S}$ is a monotone nondecreasing sequence. By hypothesis the estimate $\hat{h}$ of each node is a lower bound on h; thus we have $i_p = r$, since the $\hat{f}$ value of the goal node cannot be smaller than any previous $\hat{f}$ value.

We want to show now that $p \leqslant N$, where p is the length of the sequence $\bar{S}$. In order to do this, let us assume that a node n is closed at the ith iteration with a total estimate f_i. Since a node can be reopened only if its $\hat{g}$ value is reduced, then node n can be closed again at some subsequent step only with an $\hat{f}$ value smaller than f_i. However $\bar{S}$ is a nondecreasing sequence and therefore there cannot be two elements of $\bar{S}$ which give the $\hat{f}$ value of the same node at two different steps of the algorithm. Hence, the length of $\bar{S}$ cannot be greater than the number of nodes, i.e. $p \leqslant N$.

Given two consecutive elements of $\bar{S}$, f_{i_k} and $f_{i_{k+1}}$, let $f_j = f_{i_k}, f_{j+1}, \ldots, f_{j+m-1}, f_{j+m} = f_{i_{k+1}}$ be the elements of S which are between those two elements. According to the definition of $\bar{S}$, we have that $f_{j+l} < f_j$ $(1 \leqslant l < m)$ and $f_{j+m} \geqslant f_j$.

We want to show that no node can be closed more than once between the jth and the $(j+m)$th iteration, i.e. that $m \leqslant N$. This is easily shown since, when a node n is closed at the $(j+l)$th step $(1 \leqslant l < m)$, we know that its $\hat{g}$ value is smallest among all the nodes with $\hat{f} < f_j$. Therefore, node n can be reopened only after a node with $\hat{f} \geqslant f_j$ has been closed.

Having shown that the length of $\bar{S}$ is at most N and that between two consecutive elements of $\bar{S}$ there can be at most N elements of S, we have proved that the maximal number of iterations of algorithm B for any graph of size N is $O(N^2)$. ■

Although we have proved that in the worst case our algorithm is much better than algorithm A*, we do not know whether, in some cases, algorithm B is worse than algorithm A*. The next theorem assures us that there are no such cases, and that algorithm B is never worse than algorithm A*.

THEOREM 4.2. *Let G be any search graph with positive costs and estimates which are lower bounds on the minimal cost* ($\hat{h}(n) \leqslant h(n)$ *for each node n). Then, if algorithm*

A and algorithm* B *resolve ties in the same way, algorithm* B *does not expand more nodes than algorithm* A*.

Proof. Let $Q^B = G_1^B G_2^B \ldots G_r^B$ be the sequence of search graphs generated by algorithm B, where each graph G_i^B contains all the nodes which have been generated up to the $(i-1)$th iteration and associates with each node the $\hat{g}$ and $\hat{h}$ values and a label OPEN or CLOSED. Let $S^B = f_1 f_2 \ldots f_r$ be the sequence of the $\hat{f}$ values of the nodes in the order in which they are closed by algorithm B. Let $\bar{S}^B = f_{i_1} f_{i_2} \ldots f_{i_p}$ be the subsequence of S^B defined in the proof of the previous theorem, i.e. the steps $i_1, i_2, \ldots, i_p$ are the steps where there are no open nodes with $\hat{f} < F$. Let $\bar{Q}^B = G_{i_1} G_{i_2} \ldots G_{i_p}$ be the corresponding sequence of search graphs. Since the estimate $\hat{h}$ is a lower bound on h, we have $i_p = r$.

Let us now define in the same way the sequences $Q^A, \bar{Q}^A, S^A, \bar{S}^A$ for algorithm A*. We want to show that the subsequences of graphs $\bar{Q}^A = G_{j_1}^A G_{j_2}^A \ldots G_{j_t}^A$ and $\bar{Q}^B = G_{i_1}^B G_{i_2}^B \ldots G_{i_p}^B$ are identical, i.e. $t = p$ and $G_{j_k}^A = G_{i_k}^B$, $k = 1, \ldots, p$. This means that, if we consider the two algorithms only at the steps where there are no open nodes with $\hat{f} < F$, they behave exactly in the same way.

Of course we have $G_{j_1}^A = G_{i_1}^B$, since they both consist only of the start node. Let us assume now that we have $G_{j_k}^A = G_{i_k}^B$; we want to prove that we have $G_{j_{k+1}}^A = G_{i_{k+1}}^B$. Let n be the node selected by A* for expansion at the j_kth step. Since, by construction, at the i_kth step of algorithm B there are no open nodes with $\hat{f} < F$, and since both algorithms resolve ties in the same way, also algorithm B will select node n for expansion. It is now possible to see that algorithm A* can close between step j_k and step j_{k+1}, only and all the nodes belonging to the set V_k defined as follows:

(1) node n belongs to V_k,

(2) a node m belongs to V_k if there is a path P from n to m containing only nodes belonging to V_k and such that

$$\hat{g}_{j_k}(n) + c(P) < \hat{g}_{j_k}(m),$$

$$\hat{g}_{j_k}(n) + c(P) + \hat{h}(m) < F = \hat{g}_{j_k}(n) + \hat{h}(n),$$

where $c(P)$ is the cost of path P.

Furthermore we can see that, at the j_{k+1}th step of algorithm A*, the $\hat{g}$ value of a node m in V_k is given by $\hat{g}_{j_k}(n)$ plus the cost of the minimal cost path in V_k between n and m. But the same is true also for algorithm B, and thus we have $G_{j_{k+1}}^A = G_{i_{k+1}}^B$.

Let us now consider the number of steps of algorithm B between step i_k and i_{k+1}. Since we know that all the nodes of V_k must be closed at least once and, from the proof of the previous theorem, that no node can be closed more than once, the number of steps will be $|V_k|$. On the other hand, algorithm A* must also perform at least $|V_k|$ expansions between the steps j_k and j_{k+1}, but it can also take more, since it can close a node several times. Therefore we have shown that A* cannot run in fewer steps than B. ■

5. Conclusion

In this paper we have analyzed the complexity of algorithm A* by computing the maximum number of expansions with respect to the size of the graph to which it is applied, and we have proved that, if the estimate is not consistent, algorithm A* can have an exponential running time.

We have also given a new search algorithm, algorithm B, which has an $O(N^2)$ running time and we have shown that it never requires more steps than algorithm A*. The last result assures us that algorithm B can always be used in place of algorithm A* without having any loss in efficiency. In particular, when searching trees or graphs with a consistent estimate, both algorithms will have the same behaviour, but, with a nonconsistent estimate, algorithm B will, in general, have a much better behaviour than A*.

REFERENCES

1. Harris, L. R. The heuristic search under conditions of error. *Artificial Intelligence* **5** (Fall 1974), 217–234.
2. Hart, P., Nilsson, N. and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Systems, Man and Cybernet.* **4** (July 1968), 100–107.
3. Hart, P., Nilsson, N. and Raphael, B. Correction to "A formal basis for the heuristic determination of minimum cost paths". *SIGART Newsletter* (December 1972), 28–29.
4. Johnson, D. B. A note on Dijkstra's shortest path algorithm. *J. ACM* **20** (July 1973), 385–388.
5. Lawler, E. and Wood, D. Branch-and-bound methods: A survey. *Operations Research* **14** (July–August 1966), 699–719.
6. Martelli, A. and Montanari, U. An application of heuristically guided search to elimination processes. *Proc. International Computing Symposium*, Venice (April 1972), 59–72.
7. Martelli, A. and Montanari, U. From dynamic programming to search algorithms with functional costs. N.I. B-75/1, IEI, Pisa (1975).
8. Martelli, A. and Montanari, U. From dynamic programming to search algorithms with functional costs. *Proc. Fourth International Joint Conference on Artificial Intelligence*, Tbilisi (1975), 345–350.
9. Michie, D. and Ross, R. Experiments with the adaptive graph traverser. *Machine Intelligence* **5**, B. Meltzer and D. Michie, eds., Edinburgh Univ. Press, Edinburgh (1969), 301–318.
10. Nilsson, N. J. *Problem-solving Methods in Artificial Intelligence*. McGraw-Hill, New York (1971).
11. Nilsson, N. J. Artificial Intelligence. *Proc. IFIP Congress* 74, North-Holland, Amsterdam (1974), 778–801.
12. Pohl, I. First results on the effect of error in heuristic search. *Machine Intelligence* **5**, B. Meltzer and D. Michie, eds., Edinburgh Univ. Press, Edinburgh (1969), 219–236.
13. Pohl, I. Heuristic search viewed as path finding in a graph. *Artificial Intelligence* **1** (Fall 1970), 193–204.

Received October 1975; revised March 1976