ADMISSIBLE HEURISTIC SEARCH IN AND/OR GRAPHS

A. BAGCHI and A. MAHANTI

Indian Institute of Management, Calcutta, India

Abstract. AND/OR graphs play a major role in heuristic problem solving. Martelli and Montanari (1973) have developed an elegant algorithm, called HS, for heuristic search in AND/OR graphs, which they have used for converting decision tables to programs. It has so long been thought that HS yields minimal cost solution graphs only if the heuristic satisfies the so-called 'consistency condition'. It is shown here that the requirement that the heuristic be consistent can be relaxed to the one that the heuristic be merely admissible. This should encourage wider use of HS in applications.

1. Introduction

In the problem reduction approach to problem solving, a formulation called an AND/OR graph is often used to depict the ways in which a compound (or root) problem can be resolved into its simpler components. Frequently, a cost structure is defined on the arcs of the AND/OR graph, and the objective is to obtain a minimal cost solution subgraph, which determines an optimal way to solve the root problem Nilsson [4]. Unfortunately, it is an NP-complete proposition to find the minimal cost solution graph (see Horowitz and Sahni [1, pp. 530–532]). To circumvent this difficulty in situations of practical interest, heuristic estimates defined on the nodes of the AND/OR graph are employed to direct the search, thereby cutting down on the execution time. The heuristic estimates are normally required to satisfy a certain criterion called the admissibility condition (Nilsson [4, 5]).

Nilsson [4] presented an algorithm that yields minimal cost solutions for AND/OR trees with admissible heuristic estimates. Subsequently, Martelli and Montanari [2] gave a topdown algorithm for AND/OR graphs, a slightly modified version of which, called algorithm HS, was employed by them to convert decision tables to programs Martelli and Montanari [3]. HS is to be preferred to the simple topdown algorithm in applications, because HS examines fewer immediate predecessors of a node in its upward recomputation phase, since it only looks at ancestors along 'marked' arcs. But while Martelli and Montanari [2] showed that the topdown algorithm outputted minimal cost solution graphs for all admissible estimates, HS appeared to them and to Nilsson [5] to do so only if the heuristic estimates were consistent, a far more stringent restriction than admissibility. It is the purpose of this paper to show that HS in fact outputs minimal cost solutions for all admissible estimates. We prove the result in two stages. In Section 2, we
probe into the properties of Algorithm A, which is essentially identical to Procedure AO* of Nilsson [5]. In Section 3 we look at Algorithm B, which is the same as HS.

2. Algorithm A

An AND/OR graph $G$ is a directed graph with a special node $s$ called the start node, and a nonempty set of terminal leaf nodes $t, t_1, t_2, \ldots$. The start node $s$ represents the given problem which is to be solved, while the terminal leaf nodes correspond to subproblems with known solution. The nonterminal nodes $n, m, p, q, r, \ldots$ of $G$ are of three types: OR, AND, and nonterminal leaf. When $n$ is an OR node it can be solved in any one of a number of alternate ways, while if $n$ is an AND node then to solve $n$ we need to solve every one of its immediate successor subproblems. If $n$ is a nonterminal leaf node then it has no successors and is itself unsolvable. In the AND/OR graph shown in Fig. 1, the start node $s$

![Diagram of an AND/OR graph](image)

is an OR node, i.e. to solve $s$ either $m$ or $n$ will have to be solved, but it is not necessary to solve both. The node $m$ is an AND node, and to solve it both $r_1$ and $r_2$ must be solved. An AND node is distinguished from an OR node by drawing a line across the arcs connecting it to its successor nodes, as shown in the figure. The node $q$ is a nonterminal leaf, while $t_1$ and $t_2$ are terminal leaves. Throughout this paper we restrict ourselves, like Martelli and Montanari [2], to AND/OR graphs that are loopfree, i.e. not having (directed) paths that begin and end at the same node. However, we allow the graphs to be infinite in the following sense. Each node in the graph has finite degree, but the graph can have infinitely many nodes, i.e. there can be (directed) paths of infinite length.

Let $G$ be an AND/OR graph with start node $s$, and let $m$ be any node in $G$. A solution graph $D$ with root $m$ is a finite subgraph of $G$ with the following properties:

1. $m$ is in $D$. 
2. 
   

Fig. 1
Admissible heuristic search in AND/OR graphs

(ii) if \( n \) is an OR node in \( G \) and \( n \) is in \( D \), then exactly one of the immediate successors of \( n \) in \( G \) is in \( D \).

(iii) if \( n \) is an AND node in \( G \) and \( n \) is in \( D \), then all the immediate successors of \( n \) in \( G \) are in \( D \).

(iv) every maximal (directed) path in \( D \) ends in a terminal leaf node.

By a \textit{solution graph} is meant a solution graph with root \( s \). Fig. 2 shows two solution graphs of the AND/OR graph of Fig. 1.

![Fig. 2](image)

Since the AND/OR graph \( G \) can have a large number of nodes, it is not supplied explicitly to the search algorithm. We refer to \( G \) as the \textit{implicit graph}. The algorithm works with an \textit{explicit graph}, which initially consists of the start node \( s \). The start node is then expanded, i.e. the immediate successors of \( s \) are added to the explicit graph. At any moment the explicit graph has a number of \textit{tip nodes}, which are nodes with no successors in the explicit graph, and the search algorithm chooses one of these tip nodes for expansion. In this manner more and more nodes and arcs get added to the explicit graph, until finally it has one or more solution graphs as subgraphs. One of these solution graphs is then outputted by the search algorithm. Detailed expositions with examples can be found in Nilsson [3, 5].

The notion of a potential solution graph (p.s.g.) of an explicit graph is very similar to the notion of a solution graph of an implicit AND/OR graph. Let \( G' \) be an explicit graph with start node \( s \), and let \( m \) be any node in \( G' \). A p.s.g. \( D' \) with root \( m \) is a finite subgraph of \( G' \) with the following properties:

(i) \( m \) is in \( D' \),

(ii) if \( n \) is an OR node in \( G' \) and \( n \) is in \( D' \), then exactly one of the immediate successors of \( n \) in \( G' \) is in \( D' \).

(iii) if \( n \) is an AND node in \( G' \) and \( n \) is in \( D' \), then all of its immediate successors in \( G' \) are in \( D' \),

(iv) every maximal (directed) path in \( D' \) ends in a tip node of \( G' \).

By a p.s.g. is meant a p.s.g. with root \( s \).
We now bring in the idea of cost. Each arc \((m, n)\) in the AND/OR graph \(G\) has a finite arc cost \(c(m, n) \geq \delta > 0\), where \(\delta\) is a given small positive real number. Each node \(n\) in \(G\) also has an associated nonnegative heuristic estimate \(\hat{h}(n)\). A terminal leaf node always has a heuristic estimate of 0, while a nonterminal leaf node always has an infinite heuristic estimate. For convenience, we put \(\hat{h}(s) = 0\). This causes no loss in generality.

We define a function \(h\) on the nodes of the implicit graph \(G\) as follows:

(i) if \(n\) is a leaf node, then \(h(n) = \hat{h}(n)\) (so \(h(n)\) is 0 for terminal leaves and infinite for nonterminal leaves),

\[
h(n) = \min_{1 \leq i \leq k} [c(n, n_i) + h(n_i)].
\]

(ii) if \(n\) is an OR node with immediate successors \(n_1, n_2, \ldots, n_k\) in \(G\), then

\[
h(n) = \sum_{1 \leq i \leq k} [c(n, n_i) + h(n_i)].
\]

(iii) if \(n\) is an AND node with immediate successors \(n_1, n_2, \ldots, n_k\) in \(G\), then

\[
h(n) = \max_{1 \leq i \leq k} [c(n, n_i) + h(n_i)].
\]

In (iii) above we have used the sumcost criterion: An alternative definition of \(h\) uses the maxcost criterion, in which (iii) is replaced by (iii'), while (i) and (ii) remain unchanged:

(iii') if \(n\) is an AND node with immediate successors \(n_1, n_2, \ldots, n_k\) in \(G\), then

\[
h(n) = \max_{1 \leq i \leq k} [c(n, n_i) - c(n_i)]
\]

In this article we generally use the sumcost criterion, but all results hold for the maxcost criterion as well.

What does the value of \(h(n)\) tell us? If \(h(n)\) is infinite, then there cannot be a solution graph in \(G\) with root \(n\). If \(h(n)\) is finite, then we can think of \(h(n)\) as the 'cost' of that solution graph with root \(n\) which has lowest cost. Thus the objective of a heuristic search algorithm would be to determine \(h(s)\), the cost of a minimal cost solution graph.

Paralleling the definition of \(h\) of the nodes of the implicit graph \(G\), we define a function \(f\) on the nodes of an explicit graph \(G'\). Let \(n\) be a node in \(G'\). Then \(f(n)\) is defined just like \(h(n)\), with \(G\) and \(h\) replaced by \(G'\) and \(f\) throughout the definition. If \(n\) is a tip node of \(G'\), then \(f(n)\) is made equal to \(\hat{h}(n)\). We can then view \(f(n)\) as the 'cost' of a minimal cost p.s.g. with root \(n\) in \(G'\). The task of the search algorithm is to determine \(h(s)\) and the minimal cost solution graph. Initially, the explicit graph \(G'\) consists solely of the start node \(s\), and \(f(s) = \hat{h}(s) = 0\). Indeed, \(f(n)\) for any node \(n\) in the AND/OR graph \(G\) can be thought of as a crude first approximation to \(h(n)\). The node \(n\) must first appear in the explicit graph as a tip node (if it ever appears in the explicit graph at all), and at that time \(f(n) = \hat{h}(n)\). As the explicit graph grows, \(f(n)\) becomes a finer and finer estimate of \(h(n)\). At termination, we would like to have \(f(s) = h(s)\), as otherwise the minimal cost solution graph will not be obtained.
Looking at Figs. 1 and 2, and assuming that all arc costs are unity, we find that \( h(s) = 3 \) using either of the cost criteria, and the solution graph of Fig. 2(a) is of minimal cost. On the other hand in Fig. 2(b), \( h(m) \) is 6 by the sumcost criterion and 3 by the maxcost criterion. Note that in the computation of \( h(m) \) by the sumcost criterion the cost of the arc \((p, t_i)\) is counted twice.

We now present Algorithm A, the first of the two algorithms studied here, which is essentially identical to procedure \( \text{AO}^* \) of Nilsson [5]. Algorithm A begins by expanding the start node \( s \). At any iteration, when A expands an OR node \( n_i \), it adds the immediate successors \( n_1, n_2, \ldots, n_k \) of \( n \) to the explicit graph and marks one of the arcs \((n, n_i)\). Similarly, when A expands an AND node \( n_i \), it marks all the arcs \((n, n_i)\). Thus at each moment below each node \( n \) of the explicit graph, there is a p.s.g with root \( n \) having all its arcs marked. We call this the marked p.s.g. below \( n \). By the marked p.s.g. is meant the marked p.s.g. below \( s \).

The algorithm also uses a function \( \hat{f}_A \) which is similar to, but not identical with, the function \( f \) defined earlier.

**Algorithm A**

A1 [Initially the explicit graph \( G' \) consists solely of the start node \( s \)]. Set \( \hat{f}_A(s) \leftarrow \hat{h}(s) \). If \( s \) is a terminal leaf node, label \( s \) SOLVED.

A2 Repeat the following steps until \( s \) is labelled SOLVED. Then exit with \( \hat{f}_A(s) \) as the solution cost.

A2.1 Choose any non-leaf tip node \( n \) of the marked p.s.g. Expand \( n \), generating all of its immediate successors (if any). For each successor \( n_i \) of \( n \) not already present in \( G' \), set \( \hat{f}_A(n_i) \leftarrow \hat{h}(n_i) \). Label SOLVED any successors of \( n \) that are terminal leaves.

A2.2 Create a set \( S \) of nodes containing only \( n \).

A2.3 Repeat the following steps until \( S \) is empty.

A2.3.1 Remove from \( S \) a node \( m \) such that no descendant of \( m \) in \( G' \) occurs in \( S \).

A2.3.2 (a) if \( m \) has no successors then set \( e \) to an infinitely large value. [This case can arise only if \( m = n \).];

(b) if \( m \) has OR successors \( m_1, m_2, \ldots, m_k \) then set

\[
e \leftarrow \min_{i=1}^{k} [c(m, m_i) + \hat{f}_A(m_i)];
\]

Mark that arc \((m, m_i)\) for which the above minimum occurs. [Resolve ties arbitrarily but give preference to a SOLVED successor of \( m \)]; if \( m_i \) is SOLVED then label \( m \) SOLVED.

(c) if \( m \) has AND successors \( m_1, m_2, \ldots, m_k \) then set

\[
e \leftarrow \sum_{i=1}^{k} [c(m, m_i) + \hat{f}_A(m_i)];
\]
Mark all the arcs \((m, m_i)\) for \(1 \leq i \leq k\), and if every \(m_i\) is labelled \text{SOLVED} then label \(m\) \text{SOLVED}.

A2.3.3 \begin{align*}
\text{if } & \hat{f}_A(m) \neq e \text{ then set } \hat{f}_A(m) \leftarrow e. \\
\text{A2.3.4 } & \text{if } \hat{f}_A(m) \text{ changes in value at step A2.3.3 or } m \text{ is labelled \text{SOLVED, then add to } } S \text{ all immediate predecessors of } m \text{ along marked arcs. [Ignore predecessors of } m \text{ not connected to } m \text{ by a marked arc.]} 
\end{align*}

In step A2.3.2(c) the sumcost criterion has been used. An alternative formulation is possible using maxcost. An example of the operation of A can be found in Nilsson [5] (cf. pp. 107–108).

**Definition 1.** (i) Instant \(j\) refers to the time instant at which Algorithm A begins executing loop A2 for the \(j\)th time. So instant 1 is the time instant at which loop A2 is executed for the first time, i.e. the time instant at start.

(ii) Whenever A terminates, we use \(a\) to represent the last instant at which A reaches step A2. (At instant \(a\), A finds \(s\) \text{SOLVED} and exits.)

(iii) \(G'_i\) is the explicit graph at instant \(i\).

(iv) For each node \(n\) in \(G'_i\), \(H(n, j)\) is the marked p.s.g. below \(n\) at instant \(j\).

When \(n\) is a tip node, \(H(n, j)\) consists of the node \(n\) only.

(v) For each node \(n\) in \(G'_i\), \(\hat{f}_A(n, j)\) is the value of \(\hat{f}(n)\) at instant \(j\).

(vi) Let \(n\) be a node in an explicit graph \(G'\). By \(\hat{f}(n, G')\) we mean the value of \(\hat{f}(n)\) in \(G'\). If \(n\) is a node in \(G'_i\), then \(H(n, j)\) is well defined and can be thought of as an explicit graph. Let us define

\[
\text{Cost } H(n, j) = \hat{f}(n, H(n, j))
\]

It follows that if \(H(n, j)\) is the minimal cost p.s.g. with root \(n\) in \(G'\) then \(\text{Cost } H(n, j) = \hat{f}(n, G'_i)\), but not otherwise.

Consider the implicit graph \(G\) shown in Fig. 3. For the instants 3, 4 and 5, the explicit graphs have the forms shown in Fig. 4. Arc costs are shown against the arcs, and heuristic estimates of tip nodes are encircled. Marked arcs are crossed. It can be checked that (using sumcost):

\[
\text{Cost } H(s, 3) = \hat{f}_A(s, 3) = \hat{f}(s, G'_3) = 9,
\]

\[
\text{Cost } H(s, 4) = \hat{f}_A(s, 4) = \hat{f}(s, G'_4) = 11,
\]

\[
\text{Cost } H(s, 5) = \hat{f}_A(s, 5) = 8,
\]

\[
\hat{f}(s, G'_5) = 6.
\]

A clear distinction between \(\hat{f}\) and \(\hat{f}_A\) should be made. For a given explicit graph, the \(\hat{f}\) value at a node \(n\) gets defined without reference to any specific search algorithm, and gives the cost of the minimal cost p.s.g. with \(n\) as root. For the same explicit graph, \(\hat{f}_A(n)\) gets computed by Algorithm A, and does not necessarily
equal $f^*(r)$, as the above example shows. If the admissibility condition defined below is satisfied by the implicit graph, then $f^*_A(s)$ and $f(s)$ are equal at the termination of $A$. We prove this later, and this is an essential step in the derivation of our main result.

**Definition 2.** Let $G$ be an implicit AND/OR graph. A heuristic estimate function $h$ defined on the nodes of $G$ is

(a) **admissible** if for each node $n$ in $G$, $h(n) = h(n)$ (this implies in particular that $h(n)$ can be infinite only if $h(n)$ is infinite);

(b) **consistent** if for each node $r$ in $G$ with immediate successors $n_1, n_2, \ldots, n_k$,

(i) $h(n) \leq \min_{i=1}^k [c(n, n_i) + h(n_i)]$

whenever $n$ is an OR node,

(ii) $h(n) \leq \sum_{i=1}^k [c(n, n_i) + h(n_i)]$

whenever $n$ is an AND node (using sumcost).
The consistency condition for AND nodes using maxcost is

\[ \hat{h}(n) \leq \max_{1 \leq i \leq k} [c(n, n_i) + \hat{h}(n_i)]. \]

The consistency condition is called the monotone restriction in Nilsson [5]. Note that consistency implies admissibility.

We now study the properties of Algorithm A. Let us say that A terminates successfully if the start node s is labelled SOLVED after finitely many instants.

**Lemma 1.** If the heuristic \( \hat{h} \) is admissible, then at any instant \( j \), for each node \( n \) in \( G'_i \),

\[ f_A(n, j) \leq h(n). \]

**Proof.** The proof is similar to that of Lemma 1 of [2]. We do an induction on the instant \( j \), and for each instant \( j \) on the partial ordering of the nodes in \( G'_i \). When \( j = 1 \), \( s \) is the only node in \( G'_i \) and \( f_A(s, 1) = 0 \leq h(s) \). Now fix instant \( j \), and let the nodes of \( G'_i \) be topologically sorted in the order \( s = n_1, n_2, \ldots, n_r \). Since \( n_r \) has no successors at instant \( j \),

\[ f_A(n_r, j) = \hat{h}(n_r) \leq h(n_r) \]

by the admissibility of \( \hat{h} \). Suppose \( m = n_l \) for some \( l < r \). If \( m \) is a tip node then \( f_A(m, j) = h(m) \) as above. Otherwise, let the immediate successors of \( m \) be \( m_1, m_2, \ldots, m_k \). Then \( m_i \) is \( n_i \) for some \( i' \), \( l < i' < r \), so we may assume the lemma holds for \( m_i \) at all instants \( j' < j \). Thus

\[ f_A(m_i, j') = h(m_i) \quad \text{for all } j' < j. \]

There are two cases:

(i) \( m \) is an OR node. By the definition of \( h \)

\[ h(m) = \min_{1 \leq i \leq k} [c(m, m_i) + h(m_i)]. \]

But by Algorithm A

\[ f_A(m, j) = \min_{1 \leq i \leq k} [c(m, m_i) + f_A(m_i, j') \}], \]

where \( j' \) is the last instant at which \( f_A(m) \) has been updated.

(ii) \( m \) is an AND node. This time, assuming sumcost,

\[ h(m) = \sum_{1 \leq i \leq k} [c(m, m_i) + h(m_i)] \]

while

\[ f_A(m, j) = \sum_{1 \leq i \leq k} [c(m, m_i) + f_A(m_i, j') \}], \]

where \( j' \) is again the last instant at which \( f_A(m) \) has been updated.
Admissible heuristic search in AND/OR graphs

In either case

\[ \hat{f}_A(m, j) \leq h(m). \]

Lemma 2. At any instant \( j \), for each node \( n \) in \( G_j \),

\[ \text{Cost } H(n, j) = \hat{f}_A(n, j). \]

Proof. By induction similar to that in Lemma 1. \( \Box \)

Lemma 2 does not require the heuristic to be admissible. Note that in both the lemmas \( \hat{f}_A(n, j) \), and consequently \( h(n) \) and \( \text{Cost } H(n, j) \), can be infinite.

Lemma 3. If the implicit graph \( G \) has a solution graph, and if the heuristic \( \hat{h} \) is admissible, then Algorithm A terminates successfully.

Proof. Since \( G \) has a solution graph, \( h(s) \) is finite. By Lemmas 1 and 2, at any instant \( j \),

\[ \text{Cost } H(s, j) \leq h(s). \]

So the marked p.s.g. \( H(s, j) \) cannot have any nonterminal leaf nodes. Each node in \( G \) has finite degree, and each arc in \( G \) has a cost \( \geq \delta > 0 \), so there are only finitely many p.s.g.'s which can ever become marked p.s.g.'s. At each instant \( j \) a nonleaf tip node of the marked p.s.g. \( H(s, j) \) is expanded, so no p.s.g. can be a marked p.s.g. at two different instants. Thus if A fails to terminate then after finitely many instants there is no marked p.s.g., which is impossible. \( \Box \)

Theorem 1. If the implicit graph \( G \) has a solution graph, and if the heuristic \( \hat{h} \) is admissible, then

\[ \text{Cost } H(s, a) = h(s). \]

Proof. By Lemma 3, A terminates. Since \( H(s, a) \) is a solution graph, \( \text{Cost } H(s, a) \geq h(s) \). But by Lemmas 1 and 2, \( \text{Cost II}(s, a) \leq h(s) \). \( \Box \)

Thus A yields a minimal cost solution graph when the heuristic is admissible, not merely when the heuristic is consistent (cf. [5, p. 106]). But for admissible heuristics, while the marked p.s.g. at termination is a minimal cost solution graph the marked p.s.g. at an instant prior to termination is not necessarily a minimal cost p.s.g. Figs. 3 and 4 furnish an example. At instant 5, the marked p.s.g. has a cost of 8, while the minimal cost p.s.g. has a cost of 6. When the heuristic is not merely admissible, but also consistent, then the marked p.s.g. at every instant is also a minimal cost p.s.g. (Martelli and Montanari [2]). For the sake of completeness we include the proof of this result here.
Lemma 4. Let \( n \) be any node in an explicit graph \( G' \). Then if the heuristic is consistent, 
\[ \hat{h}(n) < \hat{f}(n, G'). \]

**Proof.** Topologically sort the nodes in \( G' \) and do an induction. \( \square \)

Lemma 5. If the heuristic is consistent, then for any two instants \( j_1 \) and \( j_2 \) where 
\[ j_1 < j_2, \] 
and for any node \( n \) in \( G'_{j_1} \),
\[ \hat{f}(n, G'_{j_1}) < \hat{f}(n, G'_{j_2}). \]

**Proof.** It is enough to consider the case \( j_2 = j_1 + 1 \). The lemma holds trivially when \( j_1 = 1 \) because \( \hat{h}(s) = 0 \). Let \( D' \) be the minimal cost p.s.g. with root \( n \) in \( G'_{j_1} \). If \( D' \) is a p.s.g. with root \( n \) in \( G'_{j_1} \), then since \( \hat{f}(n, G'_{j_1}) \) is the cost of a minimal cost p.s.g. in \( G'_{j_1} \), the lemma clearly holds. Otherwise, there must be a p.s.g. \( D'' \) with root \( n \) in \( G'_{j_1} \), such that every node and arc in \( D'' \) is also present in \( D' \). Then by Lemma 4
\[ \hat{f}(n, G'_{j_1}) \cdot \hat{f}(n, D'') \cdot \hat{f}(n, D') = \hat{f}(n, G'_{j_1}) \]
since some of the tip nodes in \( D'' \) are no longer tip nodes in \( D' \). \( \square \)

**Theorem 2.** If the heuristic is consistent, at each instant \( j \), for each node \( n \) in \( G'_{j} \),
\[ \text{Cost } H(n, j) = \hat{f}(n, G'_{j}). \]

**Proof.** The theorem clearly holds when \( j = 1 \). Moreover, the theorem always holds if \( n \) is a tip node in \( G'_{j} \). If the theorem fails for the first time at instant \( j \), it must fail at an OR node \( n \) for the first time at that instant. We can assume that for every successor \( m \) of \( n \) at every instant \( j' < j \), \( H(m, j') \) is a minimal cost p.s.g. with root \( m \) in \( G'_{j'} \), but \( H(n, j) \) is not a minimal cost p.s.g. with root \( n \) in \( G'_{j} \). Let \( n' \) be the immediate successor of \( n \) in \( H(n, j) \), and let \( j' < j \) be the last instant prior to \( j \) at which the computation of step A2.3.2 was made at \( n \). (If \( j' = j \), then the theorem cannot fail at \( n \) at instant \( j \).) Then \( \text{Cost } H(n, j') = \hat{f}(n, G'_{j'}) \) by our assumption that \( j \) is the first instant at which the theorem fails at \( n \). Again, \( \text{Cost } H(n', j) = \hat{f}(n', G'_{j}) = \text{Cost } H(n', j') = \hat{f}(n', j') \), as otherwise the computation of step A2.3.2 would have been done at \( n \) after instant \( j' \). By Lemma 5, for every other immediate successor \( m \) of \( n \), \( \hat{f}(m, G'_{j'}) = \hat{f}(m, G'_{j}) \). Hence by the definition of \( \hat{f} \),
\[ \hat{f}(n, G'_{j'}) = \hat{f}(n, G'_{j}) = \text{Cost } H(n, j') = \text{Cost } H(n, j) \]
which is a contradiction. \( \square \)

3. Algorithm B

In Algorithm A, new nodes get added to the set \( S \) whenever \( \hat{f}(m, 1) \neq e \) at step A2.3.4. When the heuristic is consistent, \( \hat{f}(m) \) can never decrease by Lemma 5. So Algorithm A can be modified to:
**Algorithm B.** Same as Algorithm A, except that

(i) A is to be replaced by B throughout the statement of the algorithm;
(ii) step A2.3.3 is to be changed to step B2.3.3 as given below

\[ \text{B2.3.3 If } f_B(m) < e \text{ then set } f_B(m) \leftarrow e. \]

Algorithm B is the same as Algorithm HS of Martelli and Montanari [3]. For consistent heuristics A and B behave identically. When the heuristic is admissible but not necessarily consistent, \( f_A(m) \) can occasionally decrease, but \( f_B(m) \) is nondecreasing. Since nodes get added to the set \( S \) less often in B than in A, we would expect B to take less time than A to execute. The question that arises is whether the marked p.s.g. at termination of B is a solution graph of minimal cost.

The proof of Lemma 1 goes through for Algorithm B, but Lemma 2 now fails to hold, because \( f_B(m) \) is not updated when \( f_B(m) > e \) at step B2.3.3. It is easy to construct examples to show that in general there can be an instant \( j \) at which for some node \( n \) in the explicit graph \( G \), \( f_B(n, j) > \text{Cost } H(n, j) \). However, a weaker version of Lemma 2 holds.

**Definition 3.** Whenever B terminates, instant \( b \) represents the last instant at which B reaches step B2. (At instant \( b \), B finds \( s \) SOLVED and exits.)

**Lemma 6.** At any instant \( j \), for each node \( n \) in \( G \),

\[ \text{Cost } H(n, j) \leq f_B(n, j). \]

**Proof.** By induction similar to that in Lemma 1. □

**Lemma 7.** If the implicit graph has a solution graph, and if the heuristic \( \hat{h} \) is admissible, then B terminates successfully.

**Proof.** Similar to the proof of Lemma 3. Use Lemma 6 instead of Lemma 2. □

**Lemma 8.** If B terminates, and if the heuristic \( \hat{h} \) is admissible, then for any node \( n \) in \( H(s, b) \)

\[ \text{Cost } H(n, h) = f_B(n, b). \]

**Proof.** By Lemma 1, \( f_B(n, b) = h(n) \). Since \( H(s, b) \) is a solution graph, and \( n \) is in \( H(s, b) \), we have \( \text{Cost } H(n, h) \geq h(n) \geq f_B(n, b) \). Now use Lemma 6. □

**Theorem 3.** If the implicit graph \( G \) has a solution graph, and if the heuristic \( \hat{h} \) is admissible, then

\[ \text{Cost } H(s, b) = h(s). \]

**Proof.** Similar to the proof of Theorem 1. Use Lemma 8. □
The top-down algorithm of Martelli and Montanari [2] also gives minimal cost solution graphs for all admissible heuristics. But in the upward recomputation phase, it looks at all predecessors of a node, while A looks only at predecessors along marked arcs. For AND/OR trees the two algorithms would not differ, since a node has exactly one immediate predecessor. But for AND/OR graphs, it is better to use A, since in general not all predecessors of a node will lie along marked arcs. B is even better than A, since B makes fewer upward computations than A, and therefore takes less time. Oddly enough, however, there exist AND/OR graphs for which B can expand more nodes than A. For example, for the graph of Fig. 5, a possible sequence of node expansions for A (using sumcost) is

\[ s n p m q_1 q_2 \]

Fig. 5

whereas for B a possible sequence is

\[ s n p m q_1 q_2 r. \]

All the results of this article hold with sumcost replaced by maxcost. Fig. 6 shows an AND/OR graph for which, using maxcost, B can expand more nodes than A. For A, a possible sequence of node expansions is

\[ s n m q r \]

while for B a possible sequence is

\[ s n m q r p. \]

This article has shown that the most satisfactory algorithm currently known for heuristic search in AND/OR graphs, viz. B, gives minimal cost solution graphs for all admissible heuristics, and not only for consistent heuristics, as previously thought. The performance of B when the heuristic is inadmissible appears to merit investigation. Can any general conclusions be drawn about the nature of the marked p.s.g. at termination of B in such a case?
References


