An Admissible and Optimal Algorithm for Searching AND/OR Graphs

C. L. Chang
and
J. R. Slagle

Heuristics Laboratory, Division of Computer Research and Technology, National Institutes of Health, Department of Health, Education and Welfare, Bethesda, Maryland 20014

Recommended by N. J. Nilsson

ABSTRACT

An AND/OR graph is a graph which represents a problem-solving process. A solution graph is a subgraph of the AND/OR graph which represents a derivation for a solution of the problem. Therefore, solving a problem can be viewed as searching for a solution graph in an AND/OR graph. A "cost" is associated with every solution graph. A minimal solution graph is a solution graph with minimal cost. In this paper, an algorithm for searching for a minimal solution graph in an AND/OR graph is described. If the "lower bound" condition is satisfied, the algorithm is guaranteed to find a minimal solution graph when one exists. Furthermore, the "optimality" of the algorithm is also proved.

Introduction

In automatic problem-solving, one is given a problem to solve, e.g., an integration to perform, a theorem to prove or a game position to analyze, etc. The usual approach [1–7] is to transform the original problem into several subproblems. Each subproblem is again converted into subproblems, and so on. This process can be easily represented by a directed graph. We consider that each node of a graph represents a problem statement. A problem and its subproblems are linked by arcs pointing from the node representing the problem to the nodes representing its subproblems. The relationship between a problem and its subproblems is stated by a Boolean function in disjunctive normal form. (We assume that no negative literal appears in the disjunctive normal form.) Every such Boolean function indicates whether or not a problem is solved if some of its subproblems are solved. The proposition N associated with a node n is the statement that the corresponding problem is solved. We shall use lower and upper cases to denote respectively a node and the proposition associated with it. Any such directed graph which represents the above problem-solving process is called an *AND/OR graph.* An AND/OR graph is shown in Fig. 1. In Fig. 1, the node **a** represents the original problem. The problem **a** is converted into three subproblems **b**, **c** and **d**. The relation between **a**, **b**, **c** and **d** is given by the Boolean function $A = B \vee CD$. This means that the problem **a** is solved if either the subproblem **b** is solved, or if the subproblems **c** and **d** are both solved. The subproblem **b** is transformed into the subproblems **e** and **f**, and is related by $B = EF$, and so on. To check how the problem **a** is related to **e**, **i** and **j**, we can make the following substitutions,

$$\begin{aligned} A &= B \vee CD \\ &= EF \vee CD && (\text{since } B = EF) \\ &= E(I \vee J) \vee CD && (\text{since } F = I \vee J) \\ &= EI \vee EJ \vee CD. \end{aligned}$$

This means that if the subproblems **e** and **i**, or **e** and **j** (or **c** and **d**) are solved, then the problem **a** is also solved. Later on, we shall call *EI*, *EJ* and *CD* implicants of *A*, i.e., *EI*, *EJ* and *CD* imply *A according to* the AND/OR graph shown in Fig. 1. The AND/OR graph shown in Fig. 1 is actually an AND/OR tree [7]. However, as discussed in [8], the AND/OR tree representation requires more space to handle *duplicate* nodes (nodes which represent the same problem) than does the AND/OR graph representation. Therefore, in this paper, we shall use the AND/OR graph representation, where every node represents a distinct problem.

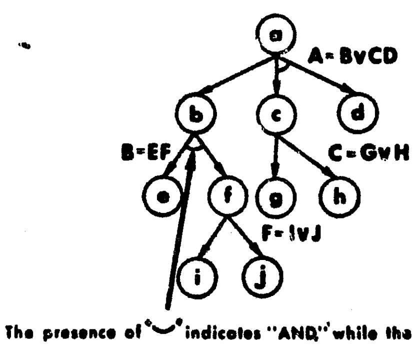

FIG. 1

Next, we consider *terminal* nodes of an AND/OR graph. The two kinds of terminal nodes are called *Type I* and *Type II* terminal nodes. A Type I terminal node represents a problem whose solution is immediately known to exist. A Type II terminal node represents a problem whose solution is immediately known not to exist. A node having no successor nodes can be considered as a Type II terminal node. In this paper, when a Type II terminal node is generated, it will be deleted from further consideration. Therefore, in the sequel, without any confusion, Type I terminal nodes will be simply called terminal nodes. A node which represents an original problem is called a *starting* node. There can be many starting nodes. If there are q starting nodes $\mathbf{s}_1, \ldots, \mathbf{s}_q$, we shall always let $S = S_1 \ldots S_q$.

We now consider the following definitions.

DEFINITION. Let $\mathbf{n}$ be a node in an AND/OR graph. Suppose $\mathbf{n}$ is related to its immediate successor nodes by a Boolean function

$$N = C_1 \vee C_2 \vee \ldots \vee C_m,$$

where C_i, $i = 1, \ldots, m$, are conjunctions of propositions. Then each C_i is called an *immediate implicant* of N.

DEFINITION. Let a conjunction $Q = N_1 \ldots N_r$, where $r \geqslant 1$. Then Q' is said to be an *immediate implicant* of Q iff Q' is a conjunction obtained from Q by replacing an N_k by one of its immediate implicants, $1 \leqslant k \leqslant r$.

DEFINITION. A conjunction Q is an *implicant* of a conjunction P iff there is a sequence of conjunctions $R_1, R_2, \ldots, R_n$ such that $P = R_1$, $Q = R_n$, and R_i is an immediate implicant of R_{i-1} for $i = 2, \ldots, n$.

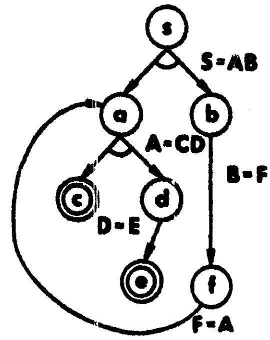

FIG. 2

DEFINITION. Let $P = N_1 \cdots N_r$. A *path graph* from the nodes $\mathbf{n}_1, \ldots, \mathbf{n}_r$ to the nodes $\mathbf{m}_1, \ldots, \mathbf{m}_s$ in an AND/OR graph G is a finite subgraph G' of G such that

(i) All the nodes $\mathbf{n}_1, \ldots, \mathbf{n}_r, \mathbf{m}_1, \ldots, \mathbf{m}_s$ are in G';

(ii) In G', only $\mathbf{n}_1, \ldots, \mathbf{n}_r$ have no arcs pointing to them, and only $\mathbf{m}_1, \ldots, \mathbf{m}_s$ have no arcs leaving from them;

(iii) For every node $\mathbf{n}$ in G' different from $\mathbf{m}_1, \ldots, \mathbf{m}_s$, there are immediate successor nodes $\mathbf{a}_1, \ldots, \mathbf{a}_t$ of $\mathbf{n}$ in G' such that $A_1 \ldots A_t$ is the only immediate implicant of N in G';

(iv) $M_1 \ldots M_s$ is an implicant of P according to G'.

DEFINITION. A path graph from the nodes $\mathbf{n}_1, \ldots, \mathbf{n}_r$ to some terminal nodes $\mathbf{t}_1, \ldots, \mathbf{t}_s$ is called a *solution graph started with* $\mathbf{n}_1, \ldots, \mathbf{n}_r$. A solution graph started with the starting nodes $\mathbf{s}_1, \ldots, \mathbf{s}_q$ will be simply called a *solution graph*.

Figure 2 shows a solution graph, where $\mathbf{s}$ is a starting node, and $\mathbf{c}$ and $\mathbf{e}$

are the terminal nodes denoted by the double circles. Since $S = AB = AF = AA = A = CD = CE$, CE is an implicant of S.
However, the graph shown in Fig. 3 is not a solution graph since CD is not an implicant of S. This graph has what is called an *impossible loop* by Slagle and Koniver [8].

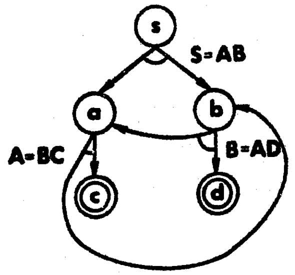

FIG. 3

In this paper, we shall associate with each arc in an AND/OR graph a cost called the *arc cost*. Let the cost of a graph be the sum of all arc costs in the graph. Therefore, every solution graph in an AND/OR graph has a cost. For any AND/OR graph, our task is to find a solution graph with minimal cost, i.e., a *minimal* solution graph. Although, with some changes if necessary, many heuristic tree or graph searching techniques [1, 9, 10, 2, 11–13, 3, 4, 14, 15, 5, 16–18, 6, 7, 19, 8] can be used in this task, we shall present another algorithm for searching for a minimal solution graph in an AND/OR graph. Our algorithm is an extension of the algorithm given by Hart et al. [20]. We shall prove that if the "lower bound" condition is satisfied, our algorithm is guaranteed to find a minimal solution graph if one exists. The optimality of our algorithm will also be discussed.

1. An Admissible Searching Algorithm

An algorithm which is guaranteed to find a minimal solution graph if one exists is called *admissible*. In this paper, we shall be concerned with AND/OR graphs implicitly specified by the starting nodes $\mathbf{s}_1, \ldots, \mathbf{s}_q$ and a (node) successor operator Γ. $1 \leqslant r \leqslant q$, application of Γ to $\mathbf{s}_r$ generates a number of successor nodes attached to $\mathbf{s}_r$ by arcs pointing from $\mathbf{s}_r$ to its successor nodes, and specifies a Boolean function relating $\mathbf{s}_r$ to its successor nodes. Application of the successor operator Γ to the successor nodes of $\mathbf{s}_r$ generates more successor nodes and Boolean functions, and so on. When a successor node of $\mathbf{n}$ is generated which is the same as a node $\mathbf{m}$ generated before, a new node is not created, but instead we provide an arc pointing from $\mathbf{n}$ to $\mathbf{m}$. Generating the successor nodes of a node by the successor operator is called *expanding* a node. A terminal node is never expanded. A node which is not a terminal node and which is not yet expanded is called an *unexpanded* node. Let a conjunction $Q = N_1 \ldots N_r$. We say that *Q is expanded* iff all the non-terminal nodes of $\mathbf{n}_1, \ldots, \mathbf{n}_r$ are expanded. When a solution graph is partially expanded, we shall call it a *partially expanded* solution graph. For example, the four graphs shown in Fig. 4 are partially expanded solution graphs of the solution graph shown in Fig. 2. We note that the conjunction of the propositions associated with all the unexpanded and terminal nodes in a partially expanded solution graph must be an implicant of S.

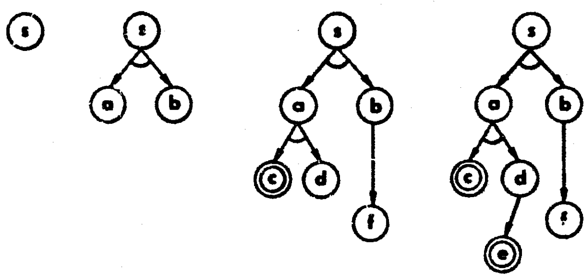

FIG. 4

Let $\mathbf{s}_1, \ldots, \mathbf{s}_q$ be the starting nodes. Our algorithm which we shall present is based upon an evaluation function $f(P)$ for each implicant P of S. This $f(P)$ can be written as $f(P) = g(P) + h(P)$, where if $P = N_1 \ldots N_r$, then $g(P)$ is the cost of a minimal path graph from $\mathbf{s}_1, \ldots, \mathbf{s}_q$ to the nodes $\mathbf{n}_1, \ldots, \mathbf{n}_r$, and $h(P)$ is the cost of a minimal solution graph started with $\mathbf{n}_1, \ldots, \mathbf{n}_r$. Note that in general if $P = N_1 \ldots N_r$, $h(P) \leqslant \sum_1^r h(N_i)$. For example, consider the minimal solution graph shown in Fig. 5(a). $P = N_1 N_2$ and $h(P) = 4$. However, if we consider $\mathbf{n}_1$ and $\mathbf{n}_2$ separately, we obtain the two solution graphs shown in Fig. 5(b). Consequently, we have $h(N_1) = 3$ and $h(N_2) = 3$. Hence,

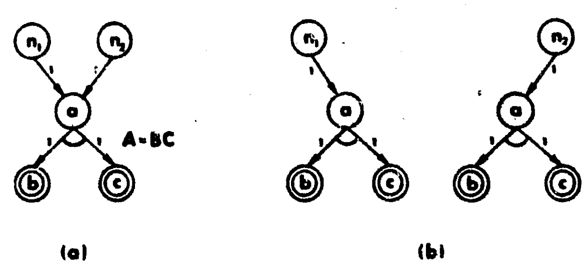

FIG. 5

$h(P) < h(N_1) + h(N_2)$. In general, $f(P)$ is not known. However, for a specific problem domain, we can use an estimate $\hat{f}(P) = \hat{g}(P) + \hat{h}(P)$ of $f(P)$, where $\hat{g}(P)$ and $\hat{h}(P)$ are the estimates of $g(P)$ and $h(P)$, respectively. In this paper, if $P = N_1 \ldots N_r$, we shall let $\hat{g}(P)$ be the cost of the path graph from the starting nodes $\mathbf{s}_1, \ldots, \mathbf{s}_q$ to the nodes $\mathbf{n}_1, \ldots, \mathbf{n}_r$ having the smallest cost so far found by the algorithm. $\hat{h}(P)$ is usually a lower bound of $h(P)$. Using this $\hat{f}(P)$, we now state our algorithm A^* as follows:

STEP 1. Let $W = \{S\}$ and R = the empty set.

STEP 2. Calculate $\hat{f}(Q)$ for each element Q in the set W. Select a P in W such that $\hat{f}(P)$ is smallest. Resolve ties arbitrarily, but always in favor of an element of W which is a conjunction of propositions associated with terminal nodes.

STEP 3. Let $P = P_1 \ldots P_r$, where P_i is the proposition associated with the node $\mathbf{p}_i$, $i = 1, \ldots, r$. If $\mathbf{p}_1, \ldots, \mathbf{p}_r$ are terminal nodes, terminate A^*; a solution graph has been found. Otherwise, go to the next step.

STEP 4. If P is expanded, go to Step 6. Otherwise, go to the next step.

STEP 5. Expand all the unexpanded non-terminal nodes of $\mathbf{p}_1, \ldots, \mathbf{p}_r$.

STEP 6. Let V be the set of all the implicants of S constructed from $P = P_1 \ldots P_r$ by replacing each (non-terminal) P_i by one of its immediate implicants, $i = 1, \ldots, r$. Let $R = R \cup \{P\}$.

STEP 7. Let $W = (W \cup V) - R$. If W is empty, terminate A^*; there is no solution graph. Otherwise, go to Step 2.

We give a simple example to illustrate the algorithm A^*. The graph to be searched is shown in Fig. 6(a), where the number beside each node $\mathbf{n}$ is the estimated cost $\hat{h}(N)$ of $h(N)$. Arc costs are assumed to be unity. For this example, if $P = N_1 \ldots N_r$, then, we define $\hat{h}(P) = (r - 1) + \text{Min}\{\hat{h}(N_1), \ldots, \hat{h}(N_r)\}$. It is clear that if $\hat{h}(N_i)$ is a lower bound for $h(N_i)$, $i = 1, \ldots, r$, then $\hat{h}(P)$ is a lower bound for $h(P)$ since for each i, a solution graph started with the nodes $\mathbf{n}_1, \ldots, \mathbf{n}_r$ contains a solution graph started with the node $\mathbf{n}_i$. In fact, $\hat{h}(N_i)$ is easier to obtain than $\hat{h}(P)$ for most practical problems. Therefore, it is often necessary to define $\hat{h}(P)$ in terms of $\hat{h}(N_i)$. In the above defined $\hat{h}(P)$, $(r - 1)$ is used since there might be a minimal solution graph started with $\mathbf{n}_1, \ldots, \mathbf{n}_r$ which consists of a solution graph started with $\mathbf{n}_k$, $1 \leqslant k \leqslant r$, and $(r - 1)$ arcs connecting from $\mathbf{n}_j$ to $\mathbf{n}_k$, $j = 1, \ldots, k - 1, k + 1, \ldots, r$. We now describe how the algorithm can be applied to obtain a minimal solution graph in the following:

(1) Expanding the node $\mathbf{s}$, we obtain the graph shown in Fig. 6(b). We know that $W = \{AB, C\}$. Since $\hat{h}(AB) = (2 - 1) + \text{Min}\{\hat{h}(A), \hat{h}(B)\} = 1 + \text{Min}\{2, 3\} = 3$, and $\hat{h}(C) = 5$, we obtain that $\hat{f}(AB) = \hat{g}(AB) + \hat{h}(AB) = 2 + 3 = 5$, and $\hat{f}(C) = \hat{g}(C) + \hat{h}(C) = 1 + 5 = 6$. $\hat{f}(AB)$ is less than $\hat{f}(C)$. Therefore, we choose AB for expansion.

(2) Expanding the nodes $\mathbf{a}$ and $\mathbf{b}$, we obtain the graph shown in Fig. 6(c). Since $AB = DE(E + F) = DEE + DEF = DE + DEF$, we have $W = \{DE, DEF, C\}$. Since $\hat{h}(DE) = (2 - 1) + \text{Min}\{\hat{h}(D), \hat{h}(E)\} = 1 + \text{Min}\{0, 1\} = 1$, and $\hat{h}(DEF) = (3 - 1) + \text{Min}\{\hat{h}(D), \hat{h}(E), \hat{h}(F)\} = 2 + \text{Min}\{0, 1, 2\} = 2$, we obtain that $\hat{f}(DE) = \hat{g}(DE) + \hat{h}(DE) = 5 + 1 = 6$, and $\hat{f}(DEF) = \hat{g}(DEF) + \hat{h}(DEF) = 5 + 2 = 7$. $\hat{f}(DE)$ and

$\hat{f}(C)$ are less than $\hat{f}(DEF)$. We can arbitrarily choose either DE or C for expansion. Suppose we choose DE for expansion.

(3) Expanding the node **e**, we obtain the graph shown in Fig. 6(d). Since $DE = DD = D$, we have $W = \{D, DEF, C\}$. Since $\hat{h}(D) = 0$, $\hat{f}(D) = \hat{g}(D) + \hat{h}(D) = 6 + 0 = 6$. Therefore, $\hat{f}(D) = \hat{f}(C) = 6$. We can arbitrarily choose D or C. However, since **d** is a terminal node, we select D. We then terminate the algorithm and obtain the solution graph shown in Fig. 6(e). The cost of this solution graph is 6 which happens to be minimal. Actually, this will always happen if $\hat{h}(Q) \leqslant h(Q)$ for all implicants Q of S.

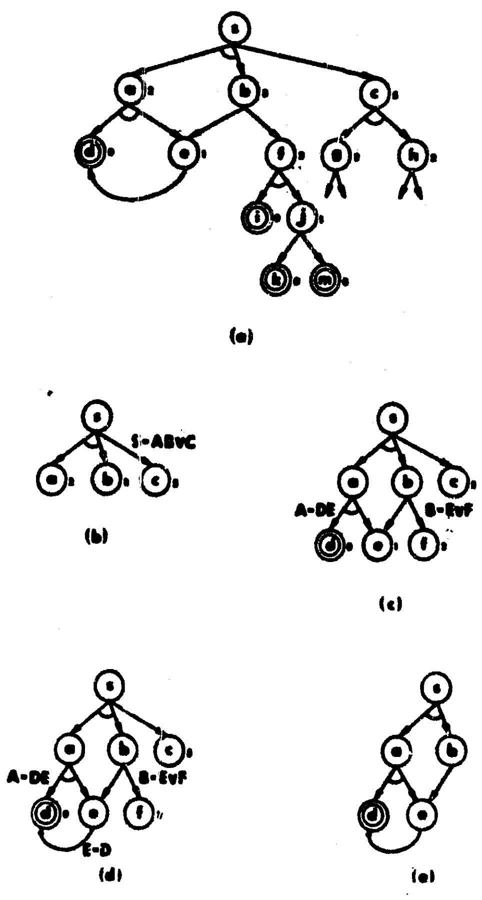

FIG. 6

An AND/OR graph is called an *OR-graph* if for each node **n** and its successor nodes $\mathbf{n}_1, \ldots, \mathbf{n}_m$ in the graph, $N = N_1 \vee \ldots \vee N_m$. Hart et al. [20] have considered an algorithm for OR-graphs. For an OR-graph, our algorithm works exactly like theirs since in this case, every implicant of S is a single proposition. For $\delta > 0$, an AND/OR graph G is called a *δ-graph* iff the cost of every arc of G is greater than or equal to δ. The following proofs of Lemma 1 and Theorem 1 are essentially those of Hart et al. [20] and Nilsson [3].

LEMMA 1. *Suppose $\hat{h}(Q) \leqslant h(Q)$ for all implicants Q of S. If P is an implicant of S selected by the algorithm A^*, and if there is a minimal solution graph, then $\hat{f}(P) \leqslant f(S)$.*

Proof. Let G_m be a minimal solution graph. Let G_0 be the partially expanded solution graph of G_m so far generated when A^* selects P. Let $\mathbf{q}_1, \ldots, \mathbf{q}_m$ be the terminal and unexpanded nodes in G_0. Then clearly, $Q = Q_1 \cdots Q_m$ is an implicant of S, and Q must be in W. Since G_m is minimal and G_0 is part of G_m, $f(S) = f(Q) = g(Q) + h(Q)$ and $\hat{g}(Q) = g(Q)$. Since $\hat{h}(Q) \leqslant h(Q)$,

$$\begin{aligned} \hat{f}(Q) &= \hat{g}(Q) + \hat{h}(Q) \\ &= g(Q) + \hat{h}(Q) \\ &\leqslant g(Q) + h(Q) \\ &= f(Q) \\ &= f(S). \end{aligned}$$

Since P is selected by A^*, $\hat{f}(P) \leqslant \hat{f}(Q)$. Therefore, $\hat{f}(P) \leqslant f(S)$. This completes the proof of Lemma 1.

We now show that A^* is admissible.

THEOREM 1. *Suppose $\hat{h}(Q) \leqslant h(Q)$ for all implicants Q of S. The algorithm A^* is admissible for all δ-graphs.*

PROOF. Assume a δ-graph G has a minimal solution graph G_m. We divide the proof into three steps as follows:

(1) A^* must terminate.

For any implicant I in W, if $I = N_1 \ldots N_r$ and if $\mathbf{n}_1, \ldots, \mathbf{n}_r$ are d arcs from the nearest node of $\mathbf{s}_1, \ldots, \mathbf{s}_q$, then $\hat{f}(I) \geqslant d\delta$. Hence, if the cost of G_m is $f(S)$, then for any such implicant $I = N_1 \ldots N_r$ with $\mathbf{n}_1, \ldots, \mathbf{n}_r$ more than $f(S)/\delta$ arcs from the nearest nodes of $\mathbf{s}_1, \ldots, \mathbf{s}_q$, $\hat{f}(I) > f(S)$. However, by Lemma 1, for any implicant P selected by A^*, $\hat{f}(P) \leqslant f(S)$. Therefore, such I will not be selected by A^*. Consequently, the algorithm A^* must eventually terminate.

(2) A^* must terminate at a solution graph.

Since G has G_m, W will not be empty at any time. Therefore, A^* will never stop in Step 7 of A^*. However, by (1), A^* must terminate. Therefore, A^* must terminate in Step 3 of A^*. This implies that a solution graph is obtained.

(3) A^* must terminate at a minimal solution graph.

Let T be the (terminal) implicant selected by A^* just before termination. By Lemma 1, $\hat{f}(T) \leqslant f(S)$. Therefore,

$$\begin{aligned} f(T) &= g(T) \\ &\leqslant \hat{g}(T) \\ &= \hat{f}(T) \\ &\leqslant f(S). \end{aligned}$$

Since $f(T)$ does not exceed $f(S)$ which is minimal, $f(T)$ must be minimal. This completes the proof of Theorem 1.

An AND/OR graph as formulated in the Introduction may be reformulated as an OR-graph: Let the starting node be S. Define the "implicant" successor operator, Γ', which, applied to S, creates a set of (say) m implicants of S. Let these m implicants of S be $S_1, \ldots, S_m$. Application of Γ' to S_i yields all the successor nodes (implicants) of S_i. Application of Γ' to S, to its successors, and so forth as long as new nodes (implicants) can be generated results in an explicit specification of an OR-graph G'. Thus, it seems that we may prove Theorem 1 by applying the theorem of Hart et al. [20] to the OR-graph G'. However, there are some complications which can arise when we have to relate the minimal path in G' to the minimal solution graph in the original AND/OR graph. Therefore, we rather use a direct proof of Theorem 1 as given above.

2. The Optimality of A*

The algorithm A^* is actually a family of algorithms; the choice of a particular function $\hat{h}$ selects a particular algorithm from the family. In this section, we shall consider how the choice of an $\hat{h}$ will effect the number of nodes being expanded by the algorithm A^*. Hart et al. [20] showed that, for OR-graphs, if a lower bound $\hat{h}$ for h used by algorithm A is greater than that used by B, then A will generally expand fewer nodes than B does. It is in this sense that we shall compare algorithms in the family of the algorithm A^*. We shall say that algorithm A_1 is *more informed* than algorithm A_2 iff $\hat{h}_1(P) > \hat{h}_2(P)$ for all implicants P of S which contain at least one proposition associated with a non-terminal node. For implicants P which contain propositions associated only with terminal nodes, we assume $\hat{h}(P) = 0$. We shall say that $\hat{h}$ is *consistent* iff $\hat{h}(Q) - \hat{h}(P) \leqslant k(Q, P)$ for any implicant Q of S and any implicant P of Q, where $k(Q, P)$ is the cost of a minimal path graph from Q to P. With these two concepts – more informedness and consistency – we can now prove a theorem about the optimality of A^*. We first prove the following lemma. The proof of Lemma 2 follows that of Nilsson [3].

LEMMA 2. *If $\hat{h}$ is consistent and if P is an implicant selected by A^*, then* $\hat{g}(P) = g(P)$.

PROOF. Suppose the contrary, i.e., suppose $\hat{g}(P) > g(P)$. Let $P = P_1 \ldots P_r$. Then, there exists some minimal path graph G_0 from the starting nodes $\mathbf{s}_1, \ldots, \mathbf{s}_q$ to $\mathbf{p}_1, \ldots, \mathbf{p}_r$. Since $\hat{g}(P) > g(P)$, G_0 is only partially expanded. Let $\mathbf{q}_1, \ldots, \mathbf{q}_m$ be all the terminal and unexpanded nodes in G_0. Clearly, $Q = Q_1 \ldots Q_m$ must be in W, and P is an implicant of Q. Since G_0 is minimal, $\hat{g}(Q) = g(Q)$. Hence, $g(P) = g(Q) + k(Q, P) = \hat{g}(Q) + k(Q, P)$. Therefore, if we assume that $\hat{g}(P) > g(P)$, then $\hat{g}(P) > \hat{g}(Q) + k(Q,P)$. Adding $\hat{h}(P)$ to both sides yields $\hat{g}(P) + \hat{h}(P) > \hat{g}(Q) + k(Q, P) + \hat{h}(P)$.

However, $\hat{h}(Q) \leqslant k(Q, P) + \hat{h}(P)$. Therefore, we obtain $\hat{g}(P) + \hat{h}(P) > \hat{g}(Q) + \hat{h}(Q)$, or $\hat{f}(P) > \hat{f}(Q)$. This means Q will be selected rather than P. Therefore, $\hat{g}(P) = g(P)$. This completes the proof of Lemma 2.

We can now prove the optimality of the algorithm A^*.

THEOREM 2. *Let A_1 and A_2 be two admissible algorithms in A^*. If A_1 is more informed than A_2, if $\hat{h}_1$ used in A_1 is consistent, and if $\hat{h}_1(Q) \leqslant h(Q)$ for any implicant Q of S, then, for any δ-graph which has a minimal solution graph, every implicant selected by A_1 is also selected by A_2.*

PROOF. Let $P_1, P_2, \ldots$ be the sequence of implicants selected by A_1. (Note that $P_1 = S$.) Suppose Theorem 2 is not true. Then there exists the first implicant in the sequence, say P_k, such that P_k is selected by A_1 but not by A_2. Since P_k is never selected by A_2, and A_2 is admissible,

$$\hat{f}_2(P_k) \geqslant f(S),$$

or

$$\hat{g}_2(P_k) + \hat{h}_2(P_k) \geqslant f(S),$$

or

$$\hat{h}_2(P_k) \geqslant f(S) - \hat{g}_2(P_k). \tag{1}$$

Since $\hat{h}_1$ is consistent, by Lemma 2, $\hat{g}_1(P_k) = g(P_k)$. That is, if $P_k = P_{k1} \ldots P_{kr}$, then when $P_1, \ldots, P_k$ in the sequence are generated there is a minimal path graph from the starting nodes $s_1, \ldots, s_q$ to $p_{k1}, \ldots, p_{kr}$. However, P_k is the first implicant in the sequence selected by A_1 but not by A_2, hence $P_1, \ldots, P_k$ are also generated by A_2. Therefore, at some stage, when $P_1, \ldots, P_k$ are generated by A_2, we have $\hat{g}_2(P_k) = g(P_k)$. Hence, (1) becomes

$$\hat{h}_2(P_k) \geqslant f(S) - g(P_k). \tag{2}$$

On the other hand, A_1 used the evaluation function

$$\hat{f}_1(P_k) = \hat{g}_1(P_k) + \hat{h}_1(P_k).$$

From Lemma 1, we know that

$$\hat{f}_1(P_k) \leqslant f(S),$$

or

$$\hat{g}_1(P_k) + \hat{h}_1(P_k) \leqslant f(S),$$

or

$$\hat{h}_1(P_k) \leqslant f(S) - \hat{g}_1(P_k). \tag{3}$$

However, as shown above, we know that $\hat{g}_1(P_k) = g(P_k)$. Therefore, (3) becomes

$$\hat{h}_1(P_k) \leqslant f(S) - g(P_k). \tag{4}$$

From (2) and (4), we obtain

$$\hat{h}_1(P_k) \leqslant \hat{h}_2(P_k).$$

This contradicts the assumption that A_1 is more informed than A_2. This completes the proof of Theorem 2.

In the above proof, the derivation of the inequality (4) follows that of Nilsson [3]. However, we use a different proof to derive the inequality (2).

Since in the algorithm A^*, the expansion of nodes follows the selection of implicants, the following is a trivial corollary of Theorem 2.

COROLLARY. *Let A_1 and A_2 be two admissible algorithms in A^*. If A_1 is more informed than A_2, if $\hat{h}_1$ used in A_1 is consistent, and if $\hat{h}_1(Q) \leqslant h(Q)$ for any implicant Q of S, then, for any δ-graph which has a minimal solution graph, every node expanded by A_1 is also expanded by A_2.*

As implied by the above corollary, the function $\hat{h}$ plays an important role in the efficiency of the algorithm A^*. When absolutely no information can be obtained from the problem domain, we may just let $\hat{h} \equiv 0$. However, in most problem domains, we do know some information. For example, consider the following problem: A telephone company decides to connect each of the cities $\mathbf{s}_1, \ldots, \mathbf{s}_q$ to the city $\mathbf{t}$. For convenience of installation, lines will be put up along the existing highways connecting the cities $\mathbf{s}_1, \ldots, \mathbf{s}_q$ and $\mathbf{t}$. Find the shortest over-all route. Fig. 7 gives an example of such a problem, where the shortest over-all route is to lay lines from $\mathbf{s}_1$ and $\mathbf{s}_2$ to $\mathbf{b}$, then $\mathbf{b}$ to $\mathbf{t}$. We note that the shortest route from $\mathbf{s}_1$ to $\mathbf{t}$ is $\mathbf{s}_1\mathbf{at}$. For this problem, we can let a node represent a city or a junction of highways, and let $\hat{h}(N)$ be the air-line distance between the city (junction) $\mathbf{n}$ and the city $\mathbf{t}$. If $P = N_1 \ldots N_r$, let $\hat{h}(P) = \text{Min}\ \{\hat{h}(N_1), \ldots, \hat{h}(N_r)\}$ for any implicant P. Using this estimate $\hat{h}$, the algorithm would still find the shortest over-all route, but would do so by expanding considerably fewer nodes than the algorithm which uses $\hat{h} \equiv 0$.

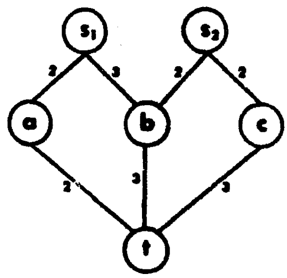

FIG. 7

We note that the algorithm A^* expands several nodes *simultaneously*. However, A^* can be adapted to expand only one node at a time. This can be easily done by replacing respectively Step 5 and Step 6 of A^* by Step 5′ and Step 6′ as follows:

STEP 5′. Among $\mathbf{p}_1, \ldots, \mathbf{p}_r$, arbitrarily select an unexpanded non-terminal node $\mathbf{p}_k$ and expand it, $1 \leqslant k \leqslant r$. (For this step, for example, we may choose $\mathbf{p}_k$ whose P_k appears most often in W, or $\mathbf{p}_k$ which has the smallest $\hat{h}(P_k)$ among $P_1, \ldots, P_r$.)

STEP 6′. Let V be the set of all the implicants of S constructed from $P = P_1 \ldots P_r$ by replacing each (non-terminal) expanded P_i by one of its immediate implicants, $i = 1, \ldots, r$. Let $R = R \cup \{P\}$.

Let B^* be the above modification of A^*. It is not difficult to see that B^* is still admissible. However, B^* is *not* optimal (in the sense stated in the above corollary) any more. In fact, B^* is a generalization of Nilsson's method [4] from AND/OR trees to AND/OR graphs.

REFERENCES

1. Amarel, S. An Approach to Heuristic Problem Solving and Theorem Proving in the Propositional Calculus, In *Systems and Computer Science*, eds. Hart, J. F., and Takasu, S. University of Toronto Press, Toronto, Ontario, Canada, 1967, pp. 125–220.
2. Ernst, G. W. and Newell, A. Generality and GPS. Doctoral dissertation at the Carnegie Institute of Technology, Pittsburgh, Pa., January, 1967.
3. Nilsson, N. J. *Problem Solving Methods in Artificial Intelligence*. McGraw-Hill, 1971.
4. Nilsson, N. J. Searching Problem-Solving and Game Playing Trees for Minimal Cost Solutions. IFIPS Congress Preprints (1968), pp. H 125–130.
5. Sandewall, E. J. Concepts and Methods for Heuristic Search. *Proc. International Joint Conference on Artificial Intelligence, Washington, D.C.*, 1969.
6. Slagle, J. R. *Heuristic Search Programs*. In *Formal Systems and Non-Numerical Problem Solving by Computers*, R. Banerji and M. D. Mesarovic (Eds.). Springer-Verlag, Berlin, 1970, pp. 246–273.
7. Slage, J. R. and Bursky, P. Experiments with a Multipurpose, Theorem-Proving Heuristic Program. *J. ACM*, **15**, No. 1 (Jan. 1968), pp. 85–99.
8. Slagle, J. R. and Koniver, D. A. Finding Resolution Proofs and Using Duplicate Goals in AND/OR Trees. To appear in *Information Sciences Journal* (1971).
9. Baylor, G. W. and Simon, H. A. Chess Mating Combinations Program. *Proceedings of the 1966 Spring Joint Computer Conference*, pp. 431–477.
10. Doran, J. E. and Michie, D. Experiments with the Graph Traverser Program. *Proc. Roy. Soc.* A, **294**, 1437 (1966), 235–259.
11. Feigenbaum, E. and Feldman, J. (Eds.) *Computer and Thought*. McGraw-Hill Book Company, New York, 1963.
12. Gelernter, H. Realization of a Geometry Proving Machine. *Proc. of the International Conference on Information Processing*, 1959. Reprinted in [11].
13. Michie, D. Strategy Building with the Graph Transver. In Collins, N. L., and Michie, D. (Eds.), *Machine Intelligence* **1**. Oliver and Boyd, Edinburgh, 1967, pp. 135–152.
14. Samuel, A. Some Studies in Machine Learning Using the Game of Checkers. *IBM J.* 3 (1959), 211–229. Reprinted in [11].
15. Samuel, A. Some Studies in Machine Learning Using the Game of Checkers II. Recent Progress. *IBM J. of Research and Development*, **11**, No. 6 (1967), pp. 601–617.
16. Sandewall, E. J. A Planning Problem Solver Based on Look-Ahead in Stochastic Game Trees. *J. ACM*, **16**, No. 3 (July 1969), pp. 364–382.
17. Slagle, J. R. A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus. Reprinted in [11].
18. Slagle, J. R. A Multipurpose Theorem-Proving, Heuristic Program that Learns. *Proc. of the IFIP Congress*, 1965.
19. Slagle, J. R. and Dixon, J. K. Experiments with Some Programs that Search Game Trees. *J. ACM*, **16**, No. 2 (April 1969), pp. 189–207.
20. Hart, P. E., Nilsson, N. J. and Raphael, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Trans. or System Sciences and Cybernetics*, Vol. SSC-4, No. 2 (July 1968), pp. 100–107.

Accepted March 9, 1971