A General Heuristic Bottom-up Procedure for Searching AND/OR Graphs

VIPIN KUMAR

Department of Computer Science,
University of Minnesota, Minneapolis, Minnesota 55455

Communicated by Laveen N. Kanal

ABSTRACT

This paper presents a general heuristic bottom-up procedure for finding a least-cost solution tree of an AND/OR graph when the cost functions associated with the arcs are monotone. Since monotone cost functions are very general, the procedure is applicable to a very large number of problems. The procedure works for both cyclic and acyclic AND/OR graphs, and subsumes most of the known bottom-up procedures for searching AND/OR graphs. Many state-space search procedures and dynamic programming procedures are also special cases of this procedure.

1. INTRODUCTION

This paper presents a general heuristic bottom-up procedure for finding a least-cost solution tree of an AND/OR graph when the cost functions associated with the arcs are monotone. Since monotone cost functions are very general, the procedure is applicable to a very large number of problems. The procedure works for both cyclic and acyclic AND/OR graphs, and subsumes most of the known bottom-up procedures for searching AND/OR graphs. Many state-space search procedures and dynamic procedures are also special cases of our procedure. The procedure develops solutions for the subproblems of an AND/OR graph (in an order determined by the heuristic information) until an optimal solution tree of the AND/OR graph is found. This framework is different from the heuristic top-down search of AND/OR graphs

This work was supported in part by Army Research Office grant #DAAG29-84-K-0060 to the Artificial Intelligence Laboratory at the University of Texas at Austin, when the author was on the faculty at the University of Texas.

(e.g., [4], AO* and related search procedures [22], [2], and [17]) and game trees (e.g., alpha-beta [22], B* [3], and SSS* [24]), in which an optimal solution tree of the AND/OR graph is found by selectively developing various possible solutions [16, 23, 14].

In principle, a least-cost solution tree of an AND/OR graph can be found by performing search in either top-down or bottom-up fashion. But depending upon the specific problem being solved, one technique may be superior to the other.[1] If the problem is such that its AND/OR graph contains only a few terminal nodes but has very many solution trees stemming from the root (e.g., problem 1.3, p. 32 in [23]), then it would be more efficient to perform the search in a bottom-up fashion. In some problems, it may be more natural to generate the graph bottom-up, as the problem reduction operators may only be known in the reverse direction. For example, the generalized state-space graphs described in [25] can be viewed as AND/OR graphs in which the problem reduction operators are described in reverse. See [25] for a discussion of problem characteristics that may influence the direction of search in an AND/OR graph.

Many breadth-first, depth-first, and heuristic strategies for conducting top-down search are already well known (e.g., AO*, alpha-beta, SSS*, B*). But all the known bottom-up search procedures to date (with the exception of algorithms in [11] and [18], which use a limited amount of problem-specific information to constrain search) are essentially breadth-first. The procedure presented in this paper provides a mechanism for using problem-specific heuristic information in the bottom-up search of AND/OR graphs. The actual amount of benefit gained is dependent upon the kind of heuristic information available and the problem domain itself. A preliminary version of this paper appeared in [13].

In Section 2 we briefly review AND/OR graphs, define a cost function on the solution trees of an AND/OR graph, and discuss the relationship between the problem of finding a least-cost solution tree of an AND/OR graph and the problems solved by dynamic programming. In Section 3 we present a general bottom-up procedure and show that a number of bottom-up procedures for searching AND/OR graphs [18, 19, 11] as well as dynamic programming procedures [1] are special cases of this procedure. In Section 4, we show that the A* algorithm for state-space search [22] can be viewed as a special case of our general bottom-up procedure. Section 5 discusses the significance of this work in the context of the author's previous work on a unified approach to search procedures.

[1]Many top-down procedures for searching AND/OR graphs (such as given in [22], [2] and [17]) use a bottom-up procedure to compute the merit/cost of partial solution trees. These procedures still search the solution-space in a top-down fashion.

2. AND/OR GRAPHS

Following the terminology in [22] and [19], we define AND/OR graphs as hypergraphs. Each node of an AND/OR graph represents a problem, and a special node root(G) called *root* of G represents the original problem to be solved. Transformation of a problem into a set of subproblems is depicted by a hyperarc directed from a parent node to a set of successor nodes. These hyperarcs are also called connectors. A hyperarc p: $n \rightarrow n_1, \ldots, n_k$ is a k-connector that shows that the problem n can be solved by solving the subproblems $n_1, \ldots, n_k$. A node having successors is called *nonterminal*. In general, a nonterminal node can have more than one hyperarc directed from it. Nodes with no successors are called *terminal*, and each terminal node represents a primitive problem.[2]

An AND/OR graph G is *cyclic* if no node of G is a successor of itself. An AND/OR graph G is called an AND/OR tree if G is acyclic and every node except root(G) has exactly one parent. Every AND/OR graph G can be unfolded (by creating duplicates of all nodes of G having multiple parents) to build an equivalent AND/OR tree called unfold(G). Note that if G is not acyclic, then unfold(G) will be an infinite structure.

Given an AND/OR graph representation of a problem, we can identify its different solutions, each one represented by a *solution tree*. A solution tree T of an AND/OR graph G is an AND/OR tree with the following properties: (i) root(G) = root(T); (ii) if a nonterminal node n of unfold(G) is in T, then exactly one hyperarc p: $n \rightarrow n_1, \ldots, n_k$ is directed from it in T, where p is one of the hyperarcs directed from n in unfold(G).

A solution tree T of G represents a plausible "problem reduction scheme" for solving the problem modeled by the root node of G. The subgraph G_n' of G rooted at a node n is in fact a problem reduction formulation of the problem represented by n, and a solution tree of G_n' represents a solution to that problem. By *a solution tree rooted at n* we mean a solution tree of G_n'. We define height(T) as the distance (in terms of number of arcs) between root(T) and a farthest terminal node of T.

Often, a cost function f is defined on the solution trees of G, and a least-cost solution tree of G is desired.[3] There are various ways in which a

[2]The assumption that each terminal node is a primitive node is made only to simplify the definition of cost (to be defined later) of solution trees. If a terminal node represents a nonprimitive problem (i.e., a problem whose solution is not known), then the cost of the node is taken to be infinite.

[3]In many problem domains, $f(T)$ denotes the merit of the solution tree T, and a largest-merit solution tree of G is desired. The discussion in this paper is applicable to such cases with obvious modifications.

cost function can be defined; the one defined below is applicable in a large number of problem domains.[4]

For a terminal node n of G, let $c(n)$ denote the cost of n, i.e., the cost of solving the problem represented by n. With each k-connector p: $n \to n_1, \ldots, n_k$ we associate a k-ary cost function $t_p(r_1, \ldots, r_k)$ which denotes the cost of solving n if n is solved by solving $n_1, \ldots, n_k$ at costs $r_1, \ldots, r_k$, respectively.

For a solution tree T, we define its cost $f(T)$ recursively as follows: if T consists only of a single node $n = \text{root}(T)$, then

$$f(T) = c(n). \tag{2.1a}$$

Otherwise, $n = \text{root}(T)$ has children $n_1, \ldots, n_k$ such that p: $n \to n_1, \ldots, n_k$ is a connector. Let $T_1, \ldots, T_k$ be the subtrees of T rooted at $n_1, \ldots, n_k$. Then

$$f(T) = t_p(f(T_1), \ldots, f(T_k)). \tag{2.1b}$$

Thus the cost of a solution tree is defined recursively as a composition of the cost of its subtrees. Figure 1 shows a cyclic AND/OR graph, associated cost functions, and the computation of the cost of one of its solution trees. We define $c^*(n)$ for nodes n of an AND/OR graph G to be the minimum of the costs of the solution trees rooted at n. Thus, $c^*(\text{root}(G))$ denotes the cost of an optimum solution tree of G. Note that if n is nonterminal, then $c^*(n)$ may be undefined, as there may be an infinite number of solution trees of decreasing costs rooted at n. A cost function $t(.,\ldots,.)$ is monotone if it is monotonically nondecreasing in each variable. For example, t_{p1}, t_{p2} and t_{p3} in Figure 1 are monotone. The following theorem gives a recursive formula for $c^*(n)$.

THEOREM 2.1. *If the cost functions $t_p(.,\ldots,.)$ are monotone and if $c^*(n)$ is defined for all nodes n of G, then for the nodes n of an AND/OR graph the following recursive equations hold.* (1) *If n is a terminal node, then $c^*(n) = c(n)$.* (2) *If n is a nonterminal node, then $c^*(n) = \min\{t_p(c^*(n_1), \ldots, c^*(n_k)) \mid p$: $n \to n_1, \ldots, n_k$ is a hyperarc directed from $n\}$.*

Proof. See [12].

Thus if the cost functions t_p are monotone, then $c^*(\text{root}(G))$, the smallest of the costs of the solution trees of G, can be found by solving the above system of equations. The procedures for solving these equations can often be easily modified to build a least-cost solution tree of G. Note that we can try to find a least-cost solution tree of G by exhaustive generation and evaluation of

[4]The definition of cost functions given here is similar to the definition of recursive weight functions given in [23]. See [23] and [16] for many practical examples.

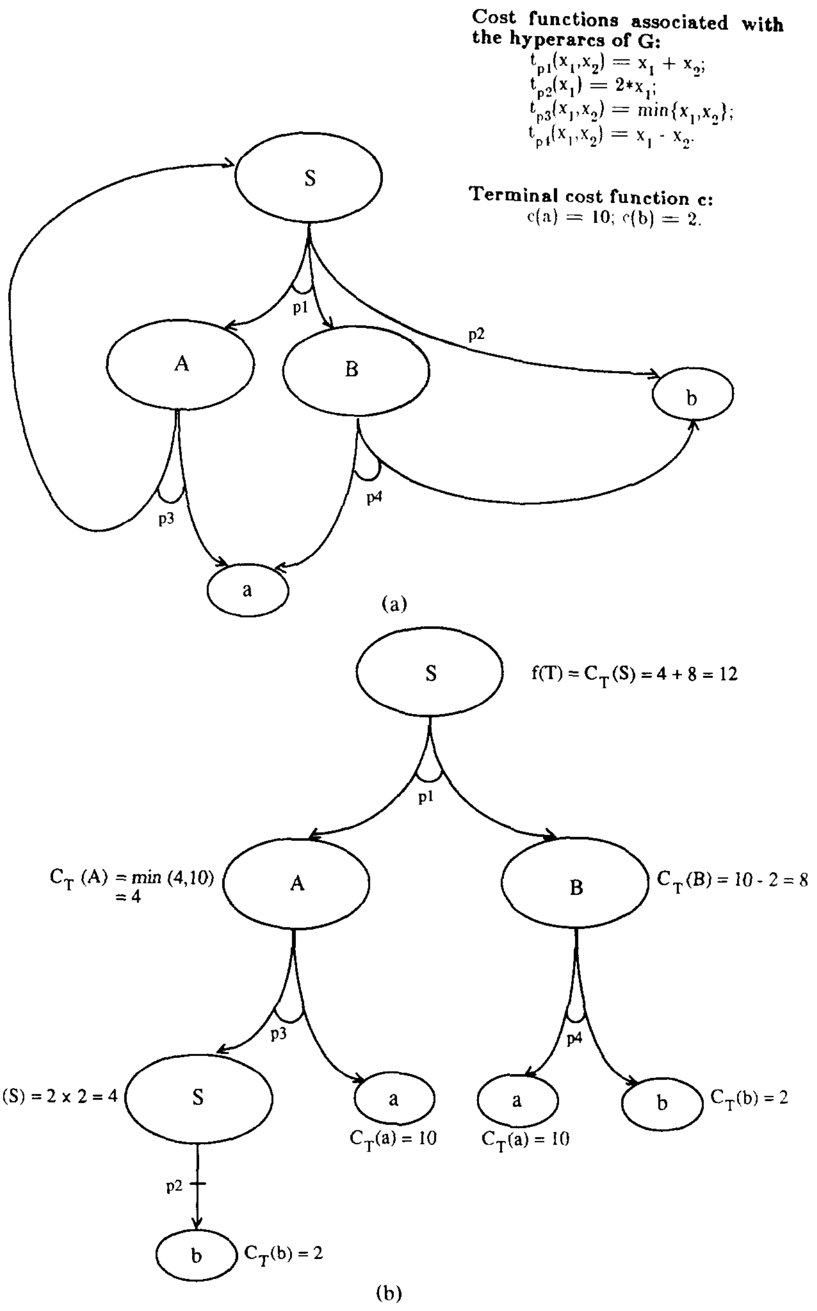
Cost functions associated with the hyperarcs of G:
$t_{p1}(x_1,x_2) = x_1 + x_2$;
$t_{p2}(x_1) = 2*x_1$;
$t_{p3}(x_1,x_2) = min\{x_1,x_2\}$;
$t_{p4}(x_1,x_2) = x_1 - x_2$.
Terminal cost function c:
c(a) = 10; c(b) = 2.
S
A
B
a
b
p1
p2
p3
p4
(a)
f(T) = C_T(S) = 4 + 8 = 12
C_T (A) = min (4,10) = 4
C_T(B) = 10 - 2 = 8
C_T(S) = 2 x 2 = 4
C_T(a) = 10
C_T(a) = 10
C_T(b) = 2
C_T(b) = 2
(b)

Fig. 1. (a) An And/Or graph G, and the associated cost functions. (b) Computation of $f(T)$ of a solution tree T of G.

all solution trees of G. But in practical problems this turns out to be far more expensive than solving the recursive equations. Of course, the exhaustive generation is not even feasible if G has an infinite number of solution trees.

RELATIONSHIP WITH DYNAMIC PROGRAMMING

Note that solving an optimization problem by Bellman's dynamic programming technique also involves converting the optimization problem into a problem of solving a set of recursive equations. Interestingly, most of the discrete optimization problems solved by dynamic programming can be formulated as the problem of finding a least-cost solution tree of an AND/OR graph with suitably defined monotone cost functions [12, 7]. We can also state a principle similar to Bellman's principle of optimality (all subpolicies of an optimum policy are also optimal). First, let us define the optimality criterion for a solution tree (the counterpart of Bellman's "policy" in our formulation). A solution tree rooted at a node n of G is called an *optimum solution tree rooted at n* if its cost is the smallest of all the solution trees rooted at n.

LEMMA 2.1. *If the cost functions t_p are monotone and if $c^*(n)$ is defined for all nodes n of G, then for every node n of G there exists an optimum solution tree rooted at n, all of whose subtrees (rooted at the immediate successors of n) are also optimal.*

Proof. See [12].

This lemma says that due to the monotonicity of $t_p(.,\dots,.)$, an optimal solution tree can always be built by optimally choosing from the alternate compositions of only the optimal subtrees. This technique of first finding the optimal solution to small problems and then using them to construct optimal solutions to successively bigger problems is at the heart of all bottom-up procedures for searching AND/OR graphs and of all dynamic programming algorithms. Hence a bottom-up search procedure for AND/OR graphs (such as the one given in the next section) can be viewed as a dynamic programming algorithm.

3. BOTTOM-UP SEARCH OF AND/OR GRAPHS

3.1. A GENERAL BOTTOM-UP SEARCH PROCEDURE

In this section we present a general bottom-up search procedure for finding an optimum solution tree of an AND/OR graph with monotone cost functions. The procedure makes use of a "lower bound" function defined as follows. If n is a node of G and x is the cost of some solution tree T rooted at

n, then $LB(n, x)$ is defined as a *lower-bound* on the cost of a solution tree of G (i.e., rooted at root(G)) and having T as a subtree; i.e., $LB(n, x) \leqslant \min\{f(T_1) | T_1$ is a solution tree of G, and T is a subtree of $T_1\}$. For a given AND/OR graph G, the following procedure finds (on terminating successfully) an optimum solution tree of G. The procedure maintains two sets of nodes called OPEN and CLOSED.

3.1.1. Procedure BUS

1. (Initialization): Initialize CLOSED to the set of terminal nodes of G. For all (terminal) nodes n in CLOSED, set $q(n) = c(n)$. Initialize OPEN with those nonterminal nodes n of G for which p: $n \to n_1, \ldots, n_k$ is a connector such that $\{n_1, \ldots, n_k\} \subseteq$ CLOSED (i.e., $n_1, \ldots, n_k$ are terminal nodes of G). For nodes n in OPEN, compute $q(n) = \min\{t_p(q(n_1), \ldots, q(n_k)) | p$: $n \to n_1, \ldots, n_k$ is a connector and $\{n_1, \ldots, n_k\} \subseteq$ CLOSED$\}$.

2. (Termination test): If root(G) is in OPEN or CLOSED and $q(\text{root}(G)) \leqslant LB(n, q(n))$ for all n in OPEN, then terminate. The cost of an optimum solution tree of G is $q(\text{root}(G))$. Otherwise, if OPEN is empty, then terminate with failure.

3a. Select and remove a node from OPEN and add it to CLOSED.

3b. For all nodes n in CLOSED, if p: $n \to n_1, \ldots, n_k$ is a connector such that $\{n_1, \ldots, n_k\} \subseteq$ CLOSED and $q(n) > t_p(q(n_1), \ldots, q(n_k))$, then recompute

$$q(n) = \min\{t_p(q(n_1), \ldots, q(n_k)) | p\colon n \to n_1, \ldots, n_k \text{ is a connector and } \{n_1, \ldots, n_k\} \subseteq \text{CLOSED}\},$$

and remove n from CLOSED and put it back in OPEN.

3c. For all nodes n in OPEN, if p: $n \to n_1, \ldots, n_k$ is a connector such that $\{n_1, \ldots, n_k\} \subseteq$ CLOSED and $q(n) > t_p(q(n_1), \ldots, q(n_k))$, then recompute

$$q(n) = \min\{t_p(q(n_1), \ldots, q(n_k)) | p\colon n \to n_1, \ldots, n_k \text{ is a connector and } \{n_1, \ldots, n_k\} \subseteq \text{CLOSED}\}.$$

3d. For all nodes n of G that are neither in OPEN nor in CLOSED, if p: $n \to n_1, \ldots, n_k$ is a connector such that $\{n_1, \ldots, n_k\} \subseteq$ CLOSED, then add n to OPEN, and compute

$$q(n) = \min\{t_p(q(n_1), \ldots, q(n_k)) | p\colon n \to n_1, \ldots, n_k \text{ is a connector and } \{n_1, \ldots, n_k\} \subseteq \text{CLOSED}\}.$$

4. Go to Step 2.

For a node n in OPEN or CLOSED, it is easily seen that $q(n)$ denotes the cost of a solution tree rooted at n such that all nodes of T are in OPEN $\bigcup$ CLOSED. The following two loop invariants are maintained by GBUS (they are true after the initialization step, and are true at the end of every execution of Step 3).

Invariant I_1. For all nodes n in OPEN or CLOSED, $q(n) \leqslant \min\{t_p(q(n_1),\ldots, q(n_k)) | p: n \to n_1,\ldots,n_k$ is a connector and $\{n_1,\ldots,n_k\} \subseteq$ CLOSED$\}$.

Invariant I_2. For all nodes n of G if $p: n \to n_1,\ldots,n_k$ and $\{n_1,\ldots,n_k\} \subseteq$ CLOSED, then n is in OPEN or CLOSED.

Lemma 3.1 follows from Invariant I_1 and the monotonicity of t_p.

LEMMA 3.1. *For a node m on OPEN or CLOSED, if T is a solution tree rooted at m such that all nodes of T (except possibly m) are in CLOSED, then* $q(m) \leqslant f(T)$.

Proof. See Appendix I.

From Lemma 3.1, it is clear that, as CLOSED contains more nodes, $q(n)$ becomes closer to $c^*(n)$. (In the limiting case, if CLOSED contains all the nodes of G, then $q(n) = c^*(n)$ for all n.) The idea behind Step 3a of BUS is (to try) to increase the size of CLOSED by transferring a node from OPEN to CLOSED. The following lemma follows from I_2 and the initialization step.

LEMMA 3.2. *Any solution tree T that has a node not in CLOSED has a subtree* T_1 *such that* $root(T_1) \in$ *OPEN and all the nodes of* T_1 *except* $root(T_1)$ *are in CLOSED.*

Proof. See Appendix I.

Lemmas 3.1 and 3.2 have the following corollary.

COROLLARY 3.1. $\min\{LB(n, q(n)) | n \in$ *OPEN*$\}$ *is a lower bound on the cost of all solution trees of G (i.e., the solution trees rooted at root(G)) that have at least one node that is not in CLOSED.*

The following theorem says that when BUS terminates successfully, $q(\text{root}(G)) = c^*(\text{root}(G))$.

THEOREM 3.1. *In BUS, when root(G) is in OPEN or CLOSED and* $q(root(G)) \leqslant LB(n, q(n))$ *for all* $n \in$ *OPEN then* $q(root(G)) = c^*(root(G))$.

Proof. Let T be a solution tree rooted at root(G) (i.e., root(G) = root(T)).

Case 1: All nodes of T are in CLOSED. Then from Lemma 1, $q(\text{root}(G)) \leqslant f(T)$.

Case 2: Otherwise, from Lemma 2, there is a subtree T_1 of T such that $\text{root}(T_1) \in$ OPEN and all nodes of T_1 except root(T_1) are in CLOSED. From Lemma 1, $f(T_1) \geqslant q(\text{root}(T_1))$.

$$\begin{aligned} f(T) &\geqslant LB(\text{root}(T_1), q(\text{root}(T_1))) \text{ (from the definition of } LB) \\ &\geqslant \min\{LB(n, q(n)) | n \in \text{OPEN}\} \ (\text{root}(T_1) \text{ is in OPEN}) \\ &\geqslant q(\text{root}(G)). \end{aligned}$$

Thus for any solution tree T rooted at root(G), $q(\text{root}(G)) \leqslant f(T)$. But $q(\text{root}(G))$ is the cost of some solution tree rooted at root(G), hence $q(\text{root}(G)) = c^*(\text{root}(G))$. ■

3.1.2. Correctness Proof of BUS

If BUS terminates unsuccessfully, then OPEN is empty and root(G) is not in CLOSED; hence, obviously G does not have any solution tree. Otherwise, if BUS terminates successfully, then from Theorem 3.1, $q(\text{root}(G)) = c^*(\text{root}(G))$. By keeping track (during the execution of BUS) of the connectors directed out of the nodes n on OPEN and CLOSED that result in the current $q(n)$ value for the node n, an optimum solution tree of G can be constructed at the successful termination of BUS.[5]

Even though upon successful termination the procedure is guaranteed to find an optimum solution tree of G, the termination itself is not guaranteed. As proved in [12], the general problem of finding an optimum solution tree of an AND/OR graph with monotone cost functions is unsolvable. The reason is that, while executing Step 3, CLOSED can grow or shrink depending upon how many nodes are transferred out of CLOSED in Step 3b. But if sufficient problem-specific information is available, termination can be guaranteed.

[5]For solution trees having cycles, it may be necessary to keep more than one "incarnation" of a node.

3.2. *USING A HEURISTIC FUNCTION TO SELECT A NODE FROM OPEN*

For a node n on OPEN, let $hf(n, x)$ denote the (heuristic) promise that a solution tree rooted at n of cost x will be a subtree of an optimum solution tree of G. If available, this information can be used to select the most promising node from OPEN in Step 4. If hf provides reasonable estimates, then the procedure can be speeded up substantially. A useful heuristic is $hf(n, x) = LB(n, x)$; because if $LB(\,.\,,\,.)$ is a tight bound, then the smaller the $LB(n, x)$ value the greater the possibility that a solution tree rooted at n of cost x is a part of an optimal solution tree of G. When $hf(n, x) = LB(n, x)$, hf is called a *lower-bound* heuristic function. If in Step 4 of BUS a node n with smallest $LB(n, q(n))$ is moved from OPEN to CLOSED, then we call it procedure BUS*.

If the lower bounds are perfect, i.e., $LB(n, x) = \min\{f(T) | T$ is a solution tree of G and has a subtree rooted at n of cost $x\}$, then $hf(n, x) = LB(n, x)$ is a perfect heuristic and BUS* will transfer only those nodes from OPEN to CLOSED that belong to an optimum solution tree of G.

The following lemma gives the condition on the lower-bound function, under which procedure BUS* can terminate whenever root(G) is transferred from OPEN to CLOSED.

LEMMA 3.3. *If $LB(root(G), x) = x$, then in BUS* whenever $root(G)$ is selected from OPEN in step $3a$, $q(root(G)) = c^*(root(G))$.*

Proof. See Appendix I.

Hence, if $LB(\text{root}(G), x) = x$, then Steps 2 and 3a of BUS* can be modified as follows:

2. If OPEN is empty, then terminate with failure.
3a. Let n be a node on OPEN such that $LB(n, q(n)) \leqslant LB(m, q(m))$ for all nodes m on OPEN. If $n = \text{root}(G)$, then terminate ($q(n)$ is the cost of an optimal solution tree of G), else remove n from OPEN and put it in CLOSED.

A lower-bound function is *logically consistent* if for all nodes n of G, $x > y \Rightarrow LB(n, x) > LB(n, y)$. A lower bound function is *heuristically consistent* if whenever T_1 is a solution tree of cost x rooted at a node n_1, and T_2 is a solution tree of cost y rooted at n_2, and T_2 is a subtree of T_1, then $LB(n_1, x) \geqslant LB(n_2, y)$. The following theorem states the condition under which a node will never be transferred from CLOSED back to OPEN (i.e., Step 2b would become superfluous in BUS*).

THEOREM 3.2. *If $LB(.\,,.)$ is both logically and heuristically consistent, then in BUS* whenever a node is selected and transferred from* OPEN *to* CLOSED, $q(n)=c^*(n)$.

Proof. Let T be a solution tree rooted at n.

Case 1: All nodes of T are in CLOSED. Then from Lemma 3.1, $q(n)\leqslant f(T)$.

Case 2: Otherwise, from Lemma 3.2, there is a subtree T_1 of T such that $\text{root}(T_1)\in$ OPEN and all nodes of T_1 except $\text{root}(T_1)$ are in CLOSED. From Lemma 3.1, $f(T_1)\geqslant q(\text{root}(T_1))$. 1. $LB(n,q(n))\leqslant LB(\text{root}(T_1),q(\text{root}(T_1)))$ (from Step 3a of BUS*) $\leqslant LB(\text{root}(T_1),f(T_1))$ (from logical consistency of LB). 2. $LB(\text{root}(T_1),f(T_1))\leqslant LB(n,f(T))$ (from heuristic consistency of LB).

From (1) and (2), $LB(n,q(n))\leqslant LB(n,f(T))$. It follows from logical consistency of LB that $q(n)\leqslant f(T)$.

Thus for any solution tree T rooted at n, $q(n)\leqslant f(T)$. But $q(n)$ is the cost of some solution tree rooted at n, hence $q(n)=c^*(n)$. ■

From Theorem 3.2, it follows that if LB is heuristically and logically consistent, then in each execution of Step 3, BUS* finds an optimal solution tree for some new nonterminal node of G. Hence, in this case, BUS* terminates in no more steps than the number of nonterminal nodes in G. If LB is a good bound, then BUS* could terminate in much fewer steps. The following lemma follows from the logical and heuristic consistency of LB.

LEMMA 3.4. *If LB is both logically and heuristically consistent, then in BUS** $\min\{LB(n,x)\mid n\in \text{OPEN}\}$ *is always nondecreasing.*

3.3. POSITIVE MONOTONE COST FUNCTIONS

A function $t(x_1,\ldots,x_k)$ is *positive monotone* if in addition to being monotone nondecreasing in each variable it satisfies the following property: $t(x_1,\ldots,x_k)\geqslant\max\{x_1,\ldots,x_k\}$. For example, t_{p1} in Figure 1 is positive monotone. For positive arguments, t_{p1} and t_{p2} are also positive monotone. If all the cost functions t_p of G are positive monotone, then it is easily seen that we can use $LB(n,x)=x$ (any solution tree containing a subtree rooted at n of cost x will have cost x or more). It follows that this lower-bound function is logically consistent and (owing to the positive monotonicity of t_p) heuristically consistent.

If the cost functions are positive monotone, then BUS* using $LB(n,q(n))=q(n)$ becomes virtually identical to Knuth's generalization of Dijkstra's algorithm [11]. For sumcost functions ($t(x_1, \ldots, x_k) = x_1 + \cdots + x_k$), BUS* using $LB(n, q(n)) = q(n)$ is identical to the heuristic bottom-up algorithm for searching AND/OR graphs by Martelli and Montanari [18].

3.4. SEARCHING ACYCLIC AND/OR GRAPHS

When G is acyclic, we can associate an integer level(n) with each node n of G such that, for any two nonterminal nodes n and m, if n is a successor of m in G, then level(n) < level(m). For each terminal node n, level(n) = 0. If level(n) is used as a heuristic for selecting a node from OPEN, then it is easily seen that (because G is acyclic) whenever a node n is transferred from OPEN to CLOSED, $q(n) = c^*(n)$. When root(G) is transferred from OPEN to CLOSED then (because of the numbering scheme used) OPEN becomes empty and the procedure terminates successfully. For sumcost functions, this procedure becomes identical to the DP algorithm given in [19].

Note that this procedure does not use a lower-bound function; hence it can be called "breadth-first." The termination is guaranteed by the acyclic nature of G. The procedure terminates after as many cycles as there are nodes in G at level below level(root(G)). This is how most of the dynamic programming procedures (with some exceptions, e.g., [20] and [10]) perform search.

If G is acyclic, then no matter what heuristic is used to select a node from OPEN in Step 2 of BUS, the procedure will terminate. The reason is that a node n is transferred from CLOSED back to OPEN only if a better solution tree rooted at n is found. Since, in an acyclic AND/OR graph, there are only a finite number of solution trees rooted at a node, a node can shuttle back and forth between OPEN and CLOSED only a finite number of times. But the complexity of the procedure can be bad, as nodes can be repeatedly transferred between OPEN and CLOSED. For many problems (e.g., a matrix multiplication problem [1]), BUS with a "random" selection strategy in Step 2 will have exponential complexity, whereas BUS using a level heuristic will have polynomial complexity.

Note that the level heuristic is effective only if the cost functions are monotone.[6] Together they guarantee that whenever a node is transferred from OPEN to CLOSED, $q(n) = c^*(n)$. Discovery of this phenomenon in certain problems led Bellman to formulate the dynamic programming paradigm. (The problems originally tackled by Bellman were limited to those described by type 3 AND/OR graphs discussed in Section 4). As pointed out in Section 2,

[6] If the cost functions are nonmonotone, then BUS using level may not terminate with an optimal solution tree.

monotonicity of cost functions and Bellman's principle of optimality are closely related.

4. RELATIONSHIP WITH STATE-SPACE SEARCH PROCEDURES

4.1. *TYPE 3 AND/OR GRAPHS*

We define *type 3* AND/OR graphs to be those AND/OR graphs that have only two types of connectors: (i) 2-connectors $n \rightarrow n_1 n_2$ such that n_1 is a nonterminal and n_2 is a terminal; (ii) 1-connectors $n \rightarrow n_1$ such that n_1 is a terminal. There is a natural correspondence between type 3 AND/OR graphs and type 3 grammars that follows from the natural correspondence between AND/OR graphs and context-free (i.e., type 2) grammars [9]. Furthermore, because of equivalence of finite state-space graphs, finite-state automata and type 3 grammars, it is possible to construct a type 3 AND/OR graph given a state-space graph, and vice versa. Figure 2 shows a state-space graph SS and its equivalent type 3 AND/OR graph G. A state M_i of SS corresponds to a node N_i in G. N_i represents the problem of getting to M_i from the start state M_1 of SS. The goal state M_4 of SS corresponds to the root node N_4 of G. For every arc in SS from M_i to M_j, there is a hyperarc $N_j \rightarrow N_i n_{i,j}$, where the terminal node $n_{i,j}$ represents the (primitive) problem of going from M_i to M_j in SS. A solution tree rooted at a node N_i corresponds to the path between the source node and M_i of SS. The source node M_1 of SS corresponds to the node N_1 of G. A special null hyperarc[7] ($N_1 \rightarrow \epsilon$) originates from N_1 in G, which denotes that the problem of going from M_1 to M_1 in SS is trivial.

4.2. *BUS* ON TYPE 3 AND/OR GRAPHS*

BUS* is essentially a generalization of the classical A* algorithm. OPEN and CLOSED in BUS* correspond to the OPEN and CLOSED lists in A*. The initialization step of BUS* corresponds to putting the start node on OPEN in A*. Step 2 corresponds to the termination check in A*. Step 3a corresponds to moving the first node n of OPEN (i.e., the node with least lower bound) to CLOSED. Step 3d corresponds to installing those children of n on OPEN that do not already exist in OPEN or CLOSED. Step 3c corresponds to possibly revising

[7]The null hyperarc in an AND/OR graph corresponds to an empty production in a context-free grammar. The existence of the null arc is assumed only to avoid a superficial difference (at the initialization step) between BUS* (working on type 3 AND/OR graphs) and A*.

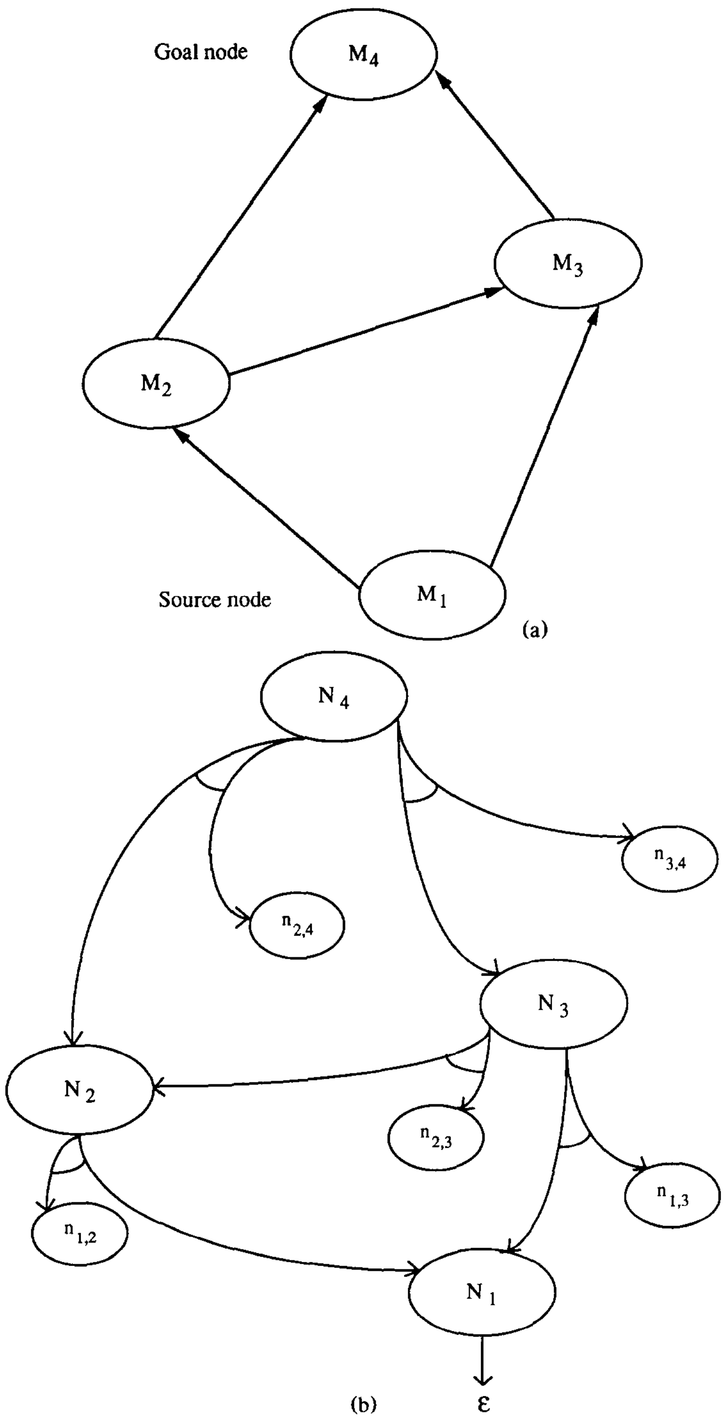

Fig. 2. (a) A state-space graph *SS*. (b) A regular And/Or graph *G* equivalent to the state-space graph *SS*.

$g(n_i)$ and $f(n_i)$ for each child n_i of n such that n_i is already on OPEN. Step 3b corresponds to possibly revising $g(n_i)$ and $f(n_i)$ for each successor n_i of n that is on CLOSED, and moving it to OPEN if necessary. Corollary 3.1 for BUS corresponds to result #2 for A* in [22].

Execution of A* on a state-space graph is identical to the execution of BUS* on a corresponding type 3 AND/OR graph whose cost functions have the following properties: (1) $t_p(x_1, x_2) = x_1 + x_2$; (2) $t_p(x_1) = x_1$; (3) $c(n) \geqslant 0$ (i.e., arc costs in the state-space graph are positive). For this case, it is possible to define $LB(n, x) = x + h(n)$, where x represents the cost of the current "type 3" solution tree rooted at n, and $h(n)$ represents the lower bound on the remaining cost (in the context of state-space graphs, x is the cost of the path from source node to n, and $h(n)$ is the lower bound on the cost of the path from n to the goal node). Since $LB(\text{root}(G), x) = x$ (because $h(\text{root}(G)) = 0$), BUS* (like A*) terminates whenever root(G) is transferred from OPEN to CLOSED. Clearly $LB(n, x)$ as defined here is logically consistent. Furthermore the heuristic consistency assumption on the lower-bound function ($LB(n, x) = x + h(n)$) is virtually identical to the so called "monotone" restriction[8] on h in [22] (which, if satisfied, guarantees that a node is never transferred back from CLOSED to OPEN). Thus, Theorem 3.2 and Lemma 3.4 can be viewed as generalizations of results #7 and #8 in [22]. Various dynamic programming procedures for finding a shortest path in a graph (e.g., [6] and [10]) are also special cases of procedure BUS for finding a least-cost solution tree of a type 3 AND/OR graph.

5. CONCLUDING REMARKS

The paradigm presented in this paper provides a general framework for using problem-specific knowledge in bottom-up search. As noted in [18], a bottom-up search algorithm generates a simpler structure than the one generated by a top-down search algorithm like AO*. At any time, for a node n, BUS only needs to keep track of one outgoing connector, which gives the current best cost solution tree rooted at n. In contrast, in AO* (or its generalizations as in [16] and [23]), all outgoing connectors have to be retained. Furthermore, the process of selecting a node in BUS (or BUS*) is much simpler than in AO*, where a graph has to be searched bottom-up to update heuristic values and top-down to select a node for expansion. Hence, for problems in which both bottom-up and top-down search are natural,

[8]Note that the monotone restriction on heuristic function h as defined in [22] has no connection with the monotonicity property of the cost functions t_p.

bottom-up search may be a better choice because of less overhead. Note that GBUS is applicable to both cyclic and acyclic AND/OR graphs, whereas AO* and its variations [2, 17] work only for acyclic graphs.

The general procedure presented in this paper was inspired by a unified approach to search procedures developed in [12], where it was shown that most of the procedures for finding an optimum solution tree of an AND/OR graph can be classified as either top-down or bottom-up. In addition to the general bottom-up procedure presented here, we have developed a general top-down search procedure for AND/OR graphs, which subsumes most of the known top-down search procedures (e.g., AO*, B*, SSS*, alpha-beta) [16]. It is natural to view the top-down search procedures for AND/OR graphs as branch-and-bound (B&B) [12, 21]. On the other hand, in the context of the general model for discrete optimization problems developed in [12] and [15], the bottom-up procedure presented in this paper can be viewed as a generalized version of dynamic programming.

Note that state-space search procedures such as A* can be considered both top-down [21] and bottom-up. The reason is that for any state-space graph, it is possible to construct two equivalent type 3 AND/OR graphs such that the top-down search in one is equivalent to the bottom-up search in the other, and vice versa [12]. This explains the confusion prevalent in the operations research literature as to whether certain shortest-path algorithms are DP or B&B. For example, Dijkstra's algorithm for shortest path [5] (an algorithm very similar to A*) has been claimed to be both DP [6] and B&B [8].

By viewing A* as a bottom-up search procedure for type 3 AND/OR graphs, we developed BUS* as a generalization of A*. (AO* can also be viewed as a generalization of A* if we view A* as a top-down search procedure for type 3 AND/OR graphs [15].) It is noteworthy that the basic structure of A* and its various properties survive two levels of generalizations: (1) in terms of cost (from sumcost to monotone cost functions); and (2) in terms of graph structure (from state-space graphs to AND/OR graphs). As discussed in Section 4.2, the steps of BUS* have almost one-to-one correspondence with the steps of A*.

APPENDIX I

LEMMA 3.1. *For a node m on* OPEN *or* CLOSED, *if T is a solution tree rooted at m such that all nodes of T (except possibly m) are in* CLOSED, *then $q(m) \leqslant f(T)$.*

Proof. By induction on the height of T.

Base case: Height$(T) = 0$, that is, T consists only of a terminal node (which is in CLOSED). $f(T) = c(\text{root}(T)) = q(\text{root}(T))$.

Induction step: Suppose the lemma holds for all solution trees of height x or less, and let T be any solution tree of height $x+1$.

Let p: $m \to m_1,\ldots,m_k$ be the connector at the root node m of T, and let $T_1,\ldots,T_k$ be subtrees of T rooted at $m_1,\ldots,m_k$. Clearly, for $1 \leqslant i \leqslant k$, height $(T_i) \leqslant x$. From the induction assumption, for $1 \leqslant i \leqslant k$, $q(m_i) \leqslant f(T_i)$.

$$q(m) \leqslant \min\{t_p(q(n_1),\ldots,q(n_k)) | p: m \to n_1,\ldots,n_k \text{ is a connector}$$

$$\text{and } \{n_1,\ldots,n_k\} \subseteq \text{CLOSED}\} \text{ (from } I_1\text{)}.$$

$$\leqslant t_p(q(m_1),\ldots,q(m_k)) \; (m_1,\ldots,m_k \text{ are in CLOSED})$$

$$\leqslant t_p(f(T_1),\ldots,f(T_k))$$

$$\text{(from the monotonicity of } t_p \text{ and the induction assumption)}$$

$$= f(T) \text{ (from the definition of } f\text{)}$$ ■

LEMMA 3.2. *Any solution tree T, which has a node not in CLOSED, has a subtree T_1 such that root$(T_1) \in$ OPEN and all the nodes of T_1 except root(T_1) are in CLOSED.*

Proof. 1. All subtrees of T height 0 (i.e., containing only a terminal node) are in CLOSED. 2. If all subtrees of T of height h or less are in CLOSED, then from Invariant I_2, roots of all subtrees of height $h+1$ are in OPEN $\cup$ CLOSED; i.e., either there exists a subtree T_1 of T such that height $(T_1) = h+1$ and root$(T_1) \in$ OPEN (and of course, all other nodes of T_1 except root(T_1) are in CLOSED) or all subtrees of height $h+1$ are in CLOSED.

Since at least one node of T is not in CLOSED, it follows from (1) and (2) that there exists a subtree T_1 of T (such that $0 <$ height$(T_1) \leqslant$ height(T) such that root$(T_1) \in$ OPEN and all nodes of T_1 except root(T_1) are in CLOSED. ■

LEMMA 3.3. *If $LB(root(G), x) = x$, then in BUS* whenever root(G) is selected from OPEN in step 3a, $q(root(G)) = c^*(root(G))$.*

Proof. If root(G) is selected in BUS* in Step 3a, then $LB(\text{root}(G), q(\text{root}(G)) = \min\{LB(n, q(n)) | n \in \text{OPEN} = q(\text{root}(G))\}$.
Hence, from Theorem 3.1, $q(\text{root}(G)) = c^*(\text{root}(G))$, and BUS* can terminate. ■

REFERENCES

1. A. Aho, J. Hopcroft, and J. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.
2. A. Bagchi and A. Mahanti, Admissible heuristic search in AND/OR graphs, *Theor. Comp. Sci.* 24:207–219 (1983).
3. H. Berliner, The B* tree search algorithm: A best-first proof procedure, *Artificial Intelligence* 12:23–40 (1979).
4. C. L. Chang and J. R. Slagle, *Artificial Intelligence* 1971.
5. E. W. Dijkstra, A note on two problems in connection with graphs, *Numer. Math.* 1:269–271 (1959).
6. S. E. Dreyfus and A. M. Law, *The Art and Theory of Dynamic Programming*, Academic Press, New York, 1977.
7. S. Gnesi, A. Martelli, and U. Montanari, Dynamic programming as graph searching, *JACM* 28:737–751 (1982).
8. P. A. V. Hall, Branch-and-bound and beyond, in *Proceedings of the Second International Joint Conference on Artificial Intelligence*, 1971, pp. 641–658.
9. P. A. V. Hall, Equivalence between AND/OR graphs and context-free grammars, *Commun. ACM* 16:444–445 (1973).
10. T. Ibaraki, Solvable classes of discrete dynamic programming, *J. Math. Anal. Appl.* 43:642–693 (1973).
11. D. E. Knuth, A generalization of Dijkstra's algorithm, *Inform. Process. Lett.* 6:1–6 (1977).
12. V. Kumar, A Unified Approach to Problem Solving Search Procedures, Ph.D. thesis, University of Maryland, 1982.
13. V. Kumar, A general bottom-up procedure for searching And/Or graphs, in *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*, Austin, Texas, 1984, pp. 182–187.
14. V. Kumar and L. Kanal, A general branch and bound formulation for understanding and synthesizing and/or tree search procedures, *Artificial Intelligence* 21(1):179–198 (1983).
15. V. Kumar and L. N. Kanal, The CDP: A unifying formulation for heuristic search, dynamic programming and branch and bound, in *Search in Artificial Intelligence* (L. N. Kanal and V. Kumar, Eds.), Springer-Verlag, 1988.
16. V. Kumar, D. S. Nau, and L. N. Kanal, A general branch-and-bound formulation for AND/OR graph and game tree search, in *Search in Artificial Intelligence* (L. N. Kanal and V. Kumar, Eds.), Springer-Verlag, 1988.
17. A. Mahanti and A. Bagchi, AND/OR graph heuristic search methods, *JACM*, pp. 28–51 (January 1985).
18. A. Martelli and U. Montanari, Additive AND/OR graphs, in *Proceedings of the Third International Joint Conference on Artificial Intelligence*, 1973, pp. 1–11.

19. A. Martelli and U. Montanari, Optimizing decision trees through heuristically guided search, *Commun. ACM* 21:1025–1039 (1978).
20. T. L. Morin and R. E. Marsten, Branch and bound strategies for dynamic programming, *Operations Res.* 24:611–627 (1976).
21. D. S. Nau, V. Kumar, and L. N. Kanal, General branch-and-bound and its relation to A* and AO*, *Artificial Intelligence* 23:29–58 (1984).
22. N. Nilsson, *Principles of Artificial Intelligence*, Tioga Publ. Co., Palo Alto, CA, 1980.
23. J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Addison-Wesley, Reading, MA, 1984.
24. G. C. Stockman, A minimax algorithm better than alpha-beta?, *Artificial Intelligence* 12:179–196 (1979).
25. G. J. VanderBrug and J. Minker, State-space problem-reduction, and theorem proving—Some relationships, *Commun. ACM* 18:107–115 (February 1975).

Received 15 April 1988; revised 24 January 1989