A General Bottom-up Procedure for Searching And/Or Graphs

Vipin Kumar
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

ABSTRACT

This paper summarizes work on a general bottom-up procedure for searching AND/OR graphs which includes a number of procedures for searching AND/OR graphs, state-space graphs, and dynamic programming procedures as its special cases. The paper concludes with comments on the significance of this work in the context of the author's unified approach to search procedures.

1. INTRODUCTION

AND/OR graphs are extensively used in such wide domains as pattern recognition, theorem proving, decision making, game playing, problem solving, and planning. A number of problems in these domains can be formulated as: "Given an AND/OR graph with certain cost functions associated with the arcs, find a least-cost solution tree of the AND/OR graph". 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 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 (e.g., AO* [20]) and game trees (e.g., alpha-beta [20], B* [1], SSS* [22]), in which an optimal solution tree of the AND/OR graph is found by selectively developing various possible solutions [9], [11]. In principle, a least-cost solution tree of an AND/OR graph may 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. 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 [8] and [15], 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.

In Section 2 we briefly review AND/OR graphs, define a cost function on the solution trees of an AND/OR graphs, 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 which includes a number of important procedures for searching AND/OR graphs and state-space graphs as well as dynamic programming procedures as its special cases. Section 4 comments upon the significance of this work in the context of author's previous work on a unified approach to search procedures.

2. AND/OR Graphs

Following the terminology in [20], [16], 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: $n \rightarrow n_1,...,n_k$ is a k-connector which shows that the problem n can be solved by solving the subproblems $n_1,...,n_k$. A Node having successors is called *nonterminal*. In general, a nonterminal node can have more than one hyperarcs directed from it. Nodes with no successors are called *terminal*, and each terminal node represents a primitive problem. An AND/OR graph G is *acyclic* 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.

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 the AND/OR graph G is in T, then exactly one hyperarc p: $n \rightarrow n_1,...,n_k$ is directed from it in T, where p is one of the hyperarcs directed from n in 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.

Often, a cost function f is defined on the solution trees of G, and a least-cost solution tree of G is desired.*** There are various ways in which a cost function can be defined; the one defined below is applicable in a large number of problem domains.

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. Let a k-ary cost function $t_p(.,...,.)$ be associated with each k-connector p: $n \rightarrow n_1,...,n_k$; $t_p(r_1,...,r_k)$ denotes the cost of solving n if n is solved by solving $n_1,...,n_k$ and if the costs of solving the nodes $n_1,...,n_k$ are $r_1,...,r_k$.

For a node n of a solution tree T, we define $c_T(n)$ to be the cost of n if the problem modeled by n is solved by the problem reduction scheme prescribed by T. It follows that

2.1a if n is a terminal node, then $c_T(n) = c(n)$;

2.1b if n is a nonterminal node and if p:$n \rightarrow n_1,...,n_k$ is the hyperarc originating from n in T, then $c_T(n) = t_p(c_T(n_1),...,c_T(n_k))$.

If T is a solution tree rooted at some node n, then we define $f(T) = c_T(n)$. Thus the cost of a solution tree is defined recursively as a composition of the cost of its subtrees. Fig. 1 shows an 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. 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. The following theorem provides a way of computing $c^*(n)$ for the nodes n of an AND/OR graph.

Theorem 2.1: If the cost functions $t_p(.,...,.)$ are monotonically nondecreasing in each variable, 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.

(i) If n is a terminal node, then
$c^*(n) = c(n)$.

(ii) If n is a nonterminal node, then
$c^*(n) = \min\{t_p(c^*(n_1),...,c^*(n_k))|$ p:$n \rightarrow n_1,...,n_k$ is a hyperarc directed from n$\}$.

Proof: See [9].

Thus if the cost functions t_p are monotone, then $c^*(root(G))$, 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.

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 [9], [4]. 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 [9].

This lemma says that due to the monotonicity of $t_p(.,...,.)$, 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.

3. 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., $lb(n,x) \leq \min\{f(T_1) \mid T_1$ 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.

***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.

Procedure BUS

(1) Initialize a set OPEN to the empty set, and a set CLOSED to the set of terminal symbols of G. For all terminal symbols n (in CLOSED), set q(n) ← c(n).

(2)a For all nodes n of G which are neither in OPEN nor in CLOSED, if $p:n \rightarrow n_1,...,n_k$ is a connector such that $\{n_1,...,n_k\} \subseteq$ CLOSED, then add n to OPEN, and compute
$q(n) = \min\{t_p(q(n_1),...,q(n_k)) |\ p:n \rightarrow n_1,...,n_k$ is a connector and $\{n_1,...,n_k\} \subseteq$ CLOSED$\}$.

(2)b For all nodes n in CLOSED, if $p:n \rightarrow n_1,...,n_k$ is a connector such that $\{n_1,...,n_k\} \subseteq$ CLOSED and $q(n) > t_p(q(n_1),...,q(n_k))$, then recompute
$q(n) = \min\{t_p(q(n_1),...,q(n_k)) |\ p:n \rightarrow n_1,...,n_k$ is a connector and $\{n_1,...,n_k\} \subseteq$ CLOSED$\}$, and remove n from CLOSED and put it back in OPEN.

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

(3) (Termination Test) If root(G) is in OPEN or CLOSED and $q(root(G)) \leq lb(n,q(n))$ for all n in OPEN, then terminate. The cost of an optimum solution tree of G is q(root(G)). Otherwise, if OPEN is empty, then terminate with failure.

(4) Select and remove a node from OPEN and add it to CLOSED.

Go to step (2)a.

The procedure maintains two sets of nodes: OPEN and CLOSED. Due to step (1) (initialization) and step (2)a, a node n of G is on CLOSED or OPEN if and only if there exists at least one solution tree rooted at n whose all other nodes are in CLOSED. For a node n on OPEN or CLOSED, it is easily seen that q(n) denotes the cost of a solution tree rooted at n. Furthermore, due to steps (2)a, b, and c (and due to the monotonicity of the cost functions t_p), for all nodes n on OPEN or CLOSED, $q(n) \leq \min\{f(T) \mid T$ is a solution tree rooted at n whose all nodes except possibly n are in CLOSED$\}$. The following theorem, needed for the correctness proof of **BUS**, is proved in [10].

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

Correctness Proof

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, $q(root(G)) = c^*(root(G))$. By keeping track (during the execution of BUS) of those connectors directed out of the nodes n on OPEN and CLOSED which 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.

Even though upon successful termination the procedure is guaranteed to find an optimum solution tree of G, but the termination itself is not guaranteed. As proved in [9], the general problem of finding an optimum solution tree of an AND/OR graph with monotone cost functions is unsolvable. But if sufficient problem-specific information is available, termination can be guaranteed.

Using Heuristic 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 smaller the lb(n,x) value 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***.

The following lemma (proved in [10]) gives the condition on lower bound, under which procedure BUS* can terminate whenever root(G) is transferred from OPEN to CLOSED.

Lemma 3.1. If lb(root(G),x) = x, then in BUS* whenever root(G) is selected from OPEN in step 4, q(root(G)) = c*(root(G)).

Hence, if lb(root(G),x) = x, then steps (3) and (4) of BUS* can be modified as follows:

(3) If OPEN is empty, then terminate with failure.

(4) Let n be a node on OPEN such that $lb(n,q(n)) \leq lb(m,q(m))$ for all nodes m on OPEN. If n = 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 $lb(n_1,x) \geq lb(n_2,y)$. The following lemma proved in [10] states the condition under which a node will never be transferred from CLOSED back to OPEN (i.e., step (2)b would become superfluous in BUS*).

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

Knuth's Generalization of Dijkstra's Algorithm

A function $t(x_1,...,x_k)$ is *positive monotone* if in addition to being monotone nondecreasing in each variable it satisfies the following property: $t(x_1,...,x_k) \geq \max\{x_1,...,x_k\}$. For example, t_{p1}, t_{p2}, t_{p3} in Fig. 1 are positive monotone. If all the cost functions t_p of G are positive monotone, then it is easily seen that we can use $lb(n,x) = x$. It follows that this lower bound function is logically consistent and (due to the positive monotonicity of t_p) heuristically consistent.

In this case BUS* becomes identical to Knuth's generalization of Dijkstra's algorithm [8]. The heuristic bottom-up algorithm for searching AND/OR graphs by Martelli and Montanari [15] is also a special case of this procedure.

Searching Acyclic AND/OR Graphs

When G is acyclic, we can number the nonterminal nodes of G such that, for any two nonterminal nodes n and m, if n is a successor of m in G, then number(n)<number(m). If number(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 (due to the numbering scheme used) OPEN becomes empty and the procedure terminates successfully.

Note that this procedure does not use a lower bound function; hence the bottom-up search is essentially uninformed (the termination is guaranteed due to the acyclic nature of G). This is how most of the dynamic programming procedures (with some exceptions, e.g., [18], [7]) perform search.

Relationship with State-Space Search Procedures

We define *regular* AND/OR graphs to be those AND/OR graphs which 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 regular AND/OR graphs and regular grammars which follows from the natural correspondence between AND/OR graphs and context-free grammars [6]. Furthermore, due to the equivalence of finite state-space graphs, finite-state automata and regular grammars, it is possible to construct a regular AND/OR graph given a state-space graph and vice versa. See Fig. 2 for a regular AND/OR graph and its equivalent state-space graph.

Note that in the context of regular AND/OR graphs, BUS* is essentially a generalization of the classical A* algorithm for state-space search. A* works on a restricted set of regular graphs in which (i) $t_p(x_1,x_2) = x_1 + x_2$; (ii) $t_p(x_1) = x_1$; (iii) $c(n) \geq 0$ (i.e., arc costs in the state-space graph are positive). Hence, it is possible to define $lb(n,x) = x + h(n)$, where x represents the cost of the current "regular" 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 $lb(root(G),x) = x$ (because $h(root(G) = 0$), 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 "monotonic" restriction*** on h in [20] (which, if satisfied. guarantees that a node is never transferred back from CLOSED to OPEN). The modifications to A* presented in [14] and [17] can also be applied to BUS* (see [10] for details).

Various dynamic programming procedures for finding a shortest path in a graph (e.g., [3], [7]) are also special cases of procedure BUS for finding a least-cost solution tree of a regular AND/OR graph (see [10]).

4. Concluding Remarks

It was discussed in [9], [12], [21] 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 [9] we presented 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). Here, we have presented a general bottom-up search procedure which subsumes most of the bottom-up procedures for searching AND/OR graphs. Almost all of the dynamic programming (DP) procedures can also be considered special cases of our bottom-up procedure. On the other hand, it is natural to view the top-down search procedures for AND/OR graphs as branch-and-bound (B&B) [9], [19].

Note that state-space search procedures like A* can be considered both top-down [19] and bottom-up. The reason is that for any state-space graph, it is possible to construct two equivalent regular AND/OR graphs such that the top-down search in one is equivalent to the bottom-up search in the other, and vice versa. 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 [2] (an algorithm very similar to A*) has been claimed to be both DP [3] and B&B [5].

***Note that the monotone restriction on heuristic function h as defined in [20] has no connection with the monotonicity property of the cost functions t_p.

Constructing a general procedure is quite useful, as it is applicable to a large number of problems and provides us insights into the nature and interrelationships of different search procedures. This is particularly important in an area which has been full of confusion. For a sample of confusing and contradictory remarks regarding the interrelationships of B&B, DP, and heuristic search procedures see [9], [12]. By identifying two natural groups of search procedures, we have resolved much of this confusion [12], [9]. Furthermore, our approach has also helped synthesize variations and parallel implementations of a number of search procedures (e.g., see [11], [13]).

REFERENCES

[1] H. Berliner, The B* Tree Search Algorithm: A Best-First Proof Procedure, *Artificial Intelligence 12,* pp. 23-40, 1979.

[2] E. W. Dijkstra, A Note on Two Problems in Connection with Graphs, *Numer. Math. 1,* pp. 269-271, 1959.

[3] S. E. Dreyfus and A. M. Law, *The Art and Theory of Dynamic Programming,* Academic Press, New York, 1977.

[4] S. Gnesi, A. Martelli, and U. Montanari, Dynamic Programming as Graph Searching, *JACM 28,* pp. 737-751, 1982.

[5] P. A. V. Hall, Branch-and-Bound and Beyond, *Proc. Second Internat. Joint Conf. on Artif. Intell.,* pp. 641-658, 1971.

[6] P. A. V. Hall, Equivalence between AND/OR Graphs and Context-Free Grammars, *Comm. ACM 16,* pp. 444-445, 1973.

[7] T. Ibaraki, Solvable Classes of Discrete Dynamic Programming, *J. Math. Analysis and Applications 43,* pp. 642-693, 1973.

[8] D. E. Knuth, A Generalization of Dijkstra's Algorithm, *Information Processing Letters 6,* pp. 1-6, 1977.

[9] V. Kumar, A Unified Approach to Problem Solving Search Procedures, Ph.D. thesis, Dept. of Computer Science, University of Maryland, College Park, December, 1982.

[10] V. Kumar, A General Heuristic Bottom up Procedure for Searching And/Or Graphs. Working paper. 1984.

[11] V. Kumar and L. Kanal, A General Branch and Bound Formulation for Understanding and Synthesizing And/Or Tree Search Procedures, *Artificial Intelligence 21,* 1, pp. 179-198, 1983.

[12] V. Kumar and L. N. Kanal, The Composite Decision Process: A Unifying Formulation for Heuristic Search, Dynamic Programming and Branch & Bound Procedures, *1983 National Conference on Artificial Intelligence (AAAI-83),* Washington, D.C., pp. 220-224, August 1983.

[13] V. Kumar and L. N. Kanal, Parallel Branch and Bound Formulations for And/Or Tree Search, *IEEE Trans. on Pattern Analysis and Machine Intelligence (to appear),* 1984.

[14] A. Martelli, On the Complexity of Admissible Search Algorithms, *Artificial Intelligence 8,* pp. 1-13, 1977.

[15] A. Martelli and U. Montanari, Additive AND/OR Graphs, *Proc. Third Internat. Joint Conf. on Artif. Intell.,* pp. 1-11, 1973.

[16] A. Martelli and U. Montanari, Optimizing Decision Trees Through Heuristically Guided Search, *Comm. ACM 21,* pp. 1025-1039, 1978.

[17] L. Mero, Some Remarks on Heuristic Search Algorithms, *IJCAI-81,* Vancouver, Canada, pp. 572-574, 1981.

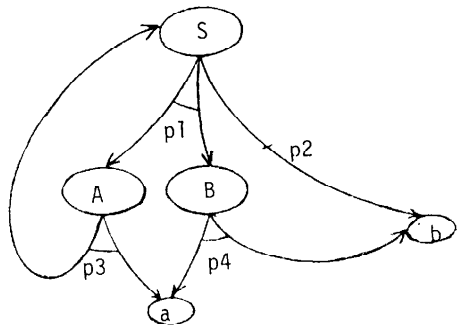
[18] T. L. Morin and R. E. Marsten, Branch and Bound Strategies for Dynamic Programming, *Operations Research 24,* pp. 611-627, 1976.

[19] D. S. Nau, V. Kumar, and L. N. Kanal, General Branch-and-Bound and its Relation to A* and AO*, *to appear in Artificial Intelligence,* 1984.

[20] N. Nilsson, *Principles of Artificial Intelligence,* Tioga Publ. Co., Palo Alto. CA, 1980.

[21] D. R. Smith, Problem Reduction Systems, Unpublished report, 1981.

[22] G. C. Stockman, A Minimax Algorithm Better than Alpha-Beta?, *Artificial Intelligence 12,* pp. 179-196, 1979.



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:

$c(a) = 10$; $c(b) = 2$.

Fig. 1(a). An And/Or graph G, and the associated cost functions.

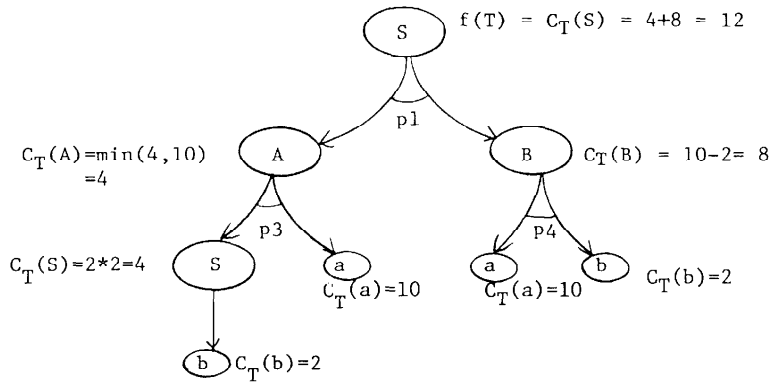


Fig. 1(b). Computation of f(T) of a solution tree T of G.

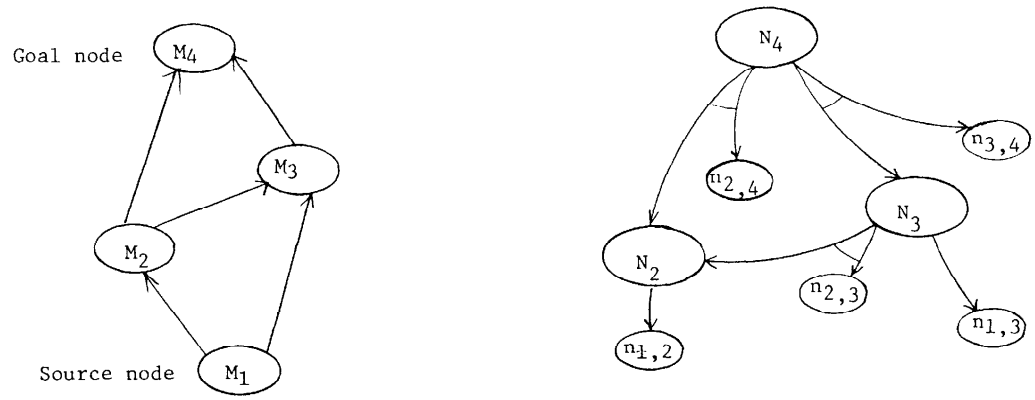


Fig. 2(a). A state space graph S.

Fig. 2(b). A regular And/Or graph G equivalent to the state space graph S. A nonterminal node N_i depicts the problem of going from the source node M_1 to node M_i in the state space graph S. A terminal node $n_{i,j}$ in G depicts the problem of going from Node M_i to M_j in S.