ADDITIVE AND/OR GRAPHs

A. Martelli and U. Montanari
Istituto di Flaborazione dell'Informazione
del Consigllo Nazionale delle Ricerche
Via s. Maria, 46 - 56100 - Pisa Italy.

Abstract

Additive AND/OR graphs are defined as AND/ /ORgraphs without circuits, which can be considered as folded AND/OR trees; i.e. the cost of a common subproblem is added to the cost as many times as the subproblem occurs, but it is computed only once. Additive AND/OR graphs are naturally obtained by reinterpreting the dynamic programming method in the light of the problem-reduction approach. An example of this reduction is given.

A top-down and a bottom-up method are proposed for searching additive AND/OR graphs. These methods are, respectively, extensions of the "arrow" method proposed by Nilsson for searching AND/OR trees and Dijkstra's algorithm for finding the shortest path. A proof is given that the two methods find an optimal solution whenever a solution exists.

1) introduction

In the literature on artificial intelligence, AND/OR trees have proved to be a good formalism for representing the problem-reduction approach to problem solving . Usually, the search is for any solution tree, but in a paper by Nilsson the problem is presented of finding the best solution tree, where arcs have a given cost, and the cost of a tree is simply the sum of the costs of the arcs. Nilsson gives there an algorithm which assumes available, for each node, an estimate of the cost of the "optimal solution tree rooted at that node.

An algorithm for searching AND/OR graphs has been proposed by Chang and Slagle[3]. Here a solution graph is defined in the usual way, and, as for trees, the cost of a solution graph is the sum of the costs of its arcs.

In this paper we introduce a new type of AND/OR graphs called additive, which can be considered as folded AND/OR trees, i.e. trees where different nodes have been recognized to be roots of equal subtrees and have been identified, thus generating AND/OR graphs without circuits. However, the cost of a solution subgraph is defined as equal to the cost of the unfolded equivalent tree. In other words, the cost of a common subproblem is added to the cost as many times as the subproblem occurs, but it is computed only once.

Additive AND/OR graphs are naturally obtained by reinterpreting the dynamic programming method of optimization in the light of the problem-reduction approach[4].

For giving an idea of the technique of reduction, we will consider here the well--known problem of halancing binary search trees, solved by Knuth viith a (modified) dynamic programming algorithm". Here a number of ordered items (lexicographically ordered words, for instance) are given, together with probabilities of a new item occurence to be any of the given items or to be a new item located in any intermediate position. For example, the data

(1) 3 [7] begin 2 [3] do 3 [5] if 8

mean that there are 3 probabilities out of 31 that a new word will be "do" and 3 nrohahilities that a new word will be any word between "do" and "if".

A binary search tree for these data is a tree of the type shown in Fig. 1. For instance if the new word "array" is generated, its proper location is found by means of three tests, namely comparisons with the given words "if" and "begin" respectively and a termination test.

Given a search tree, the average number of tests M necessary for reaching a node, is then given by summing up the products of the number mj. of tests required for any node i and its probability p_1

$$M = \sum_{i=1}^{n} m_i p_i$$

For instance, the average number of tests for the tree in Fig. 1 is

$$M = \frac{1}{31}(3\cdot 3 + 2\cdot 7 + 4\cdot 2 + 3\cdot 3 + 4\cdot 3 + 5 + 2\cdot 8) = \frac{73}{31}$$

The problem is to find a search tree with minimal cost M. For instance, the tree of Fig.1 is optimal for the data (1).

An equivalent formulation of the same problem considers frequencies instead of probabilities. In this case the cost is called weighted path length.

Two properties allow the use of a dynamic programming technique in this case. First, if T = (a A 8) is an optimal tree rooted in A and having a and 6 as subtrees, then both o and B are optimal. Furthermore, the data for a and S are disjoint substrings of the data for T. Second, if f_A ,fa ,f_B 'La *LB are the* frequencies of A,o and 8 and the weighted

path lengths of α and β, then the value

(2) $L = L_\alpha + f_\alpha + L_\beta + f_\beta + f_A$

is the weighted path length of T. For instance the left and right subtrees of the tree in Fig. 1 are the optimal trees for the data

3 [7] 2 [3] 3

and

8

Furthermore, we have $L_\alpha = 34, f_\alpha = 18, L_\beta = 8, f_\beta = 8, f_A = 5$ and $L = 73$.

In the dynamic programming algorithm, all the subproblems of the given problem are considered, whose data are substrings of the given data. Such problems are divided in lev els, according to the length of the data, and solved in increasing order* A problem at a level can be solved by picking up a root in all possible ways(*) (say JO and thus decomposing the problem in k pairs of subproblems at lower levels. The cost of every decomposition is computed using (2)and a best decomposition is chosen.

The optimal search tree problem described above is a good example of the gener al case where, at each stage, the computation of each alternative requires the sum of the costs of one or more (in fact, two) sub problems.

In this case, the structure of the problem is conveniently reflected into an additive AND/OR graph, whose AND nodes correspond to subproblem cost sums, and OR nodes to alternative selections. For instance, the AND/OR graph for the optimal search tree problem, with data (1), is shown in Fig. *2.* There,AND nodes are marked with circles and OR nodes (corresponding to subproblems) with squares. The optimal solution graph, corresponding to the search tree in Fig. 1, is blackened.

Reduction of dynamic programming to addit1 ve AND/OR graphs can imply several advantages. First, the AND/OR graph expresses the structure of the problem in the form of a partial ordering of subproblems. Dynamic pro gramming solves all the subproblems bottom--up in some static order which respects the partial orderings. However,in Section 4 of this paper a bottom-up algorithm is described, which solves every time the cheapest available subproblem, thus considering, in general, only a subset of subproblems in a dynamic order.

Second, in many cases estimates of the subproblem costs are available, which can be used for directing a top-down search. In fact, in Section 3 we give an extension of NIlsson's tree algorithm to the additive graph case, which, if the estimate is a lower bound of the minimal cost, is guaranteed to find the optimal solution graph. The algorithm is slightly simplified if the estimate satisfies a "consistency" constraint. Finally, various known heuristic techniques can be applied to additive AND/OR graphs,which can find a good solution where the exact dynamic programming algorithm is too Expensive[6].

We emphasize that this paper extends to general dynamic programming a reduction technique which is well-known in the so called "sequential" case[7]. There, each alternative can be computed by summing a constant to the cost of one simpler subproblem, thus reducing an additive AND/OR graph to an OR graph. In this case, a solution tree reduces to a path and thus shortest path algorithms apply, like Dijkstra's algorithm[8] in the uninformed case and the algorithm by Hart, Nilsson, and Raphael[9] if an estimate is available.

Finally, note that in the shortest path case no distinction between top-down and bottom-up is needed, while it is suggestive in the general case.

2) Additive AND/OR Graphs

In this paper, we shall consider AND/OR graphs without cycles. An example is given in Fig. 3. Node A is called the start node and represents the problem to be solved. Node A is called an OR node,(*) because a solution is constructed by selecting either its successor B or its successor C. Node B is called an AND node, because all of the suboroblems represent ed by its successors D and E must be solved in order to solve problem B. An AND node is indicate by a line across the arcs connecting it to the successor nodes. Nodes G and H are called terminal nodes and correspond to problems with known solution.

We assume that all the nonterminal nodes are either OR nodes or AND nodes, that is, there are no nodes corresponding to problems which can be solved by solving some of their successors.

(*) Actually, Knuth[5] gives a modified dynamic programming algorithm which excludes a priori most of the decompositions, using a particular monotonieity property.

We give here a definition of OR nodes and AND nodes, which is opposite to the one given by Nilsson[1]. We do so, because we want each node to be either an OR node or an AND node, instead, with NIlsson's definition, we could have a node which is at the same time an OR node because of one parent and an AND node because of another parent.

We shall be concerned with AND/OR graphs -implicitly specified by a start node s and a successor operator T.Application of r to any node n qenerates a finite number of successors of n and a label snecifyino whether the successors are AND nodes or OR nodes,

A solution graph of an AND/OR granh G with start node s is any suhoranh of P containing s and havino the following properties;

(1) Suppose node n of P is an OR node and is included in the solution oraph. Then one and only one of the successors of n is also included in the solution graph.

(2) Suppose node n of R is an AND node and is included in the solution graph. Then all of the successors of n are also included in the solution oranh.

(3) The solution granh is finite, meanlno that it ends in a set of terminal nodes,

Fig. 4 shows two solution oranhs of the AND/OR qraph in Fig. 3.

In general, a cost is associated with every arc of an AND/OR Grph. Let the function c(n ,n.) give the cost to he associated with the are connecting node n_i with one of its successors n_j. For additive AND/OR oranhs the cost of a solution granh is recursively defined as follows:

(1) The cost c * 0 is associated with everv terminal node in the solution oranh.

(2) Let $c_1, —, c_k$ be the costs associated with the k successors of node n in the solution oranh. Then we associate with n the cost c oiven by(*)

$$C = \sum_{i=1}^{k} (c_i + c(n,n_i))$$

(3) The cost associated with the start node s is the cost of the solution oranh.

The cost of a solution oranh can alwavs be obtained with the above definition, because solution e/raphs do not contain cvcles.

The cost of a solution graph can also be obtained by representing the solution nranh as a tree by dunlicatinq the suboranhs rooted at nodes with more than one incident arc and taking the sum of all the arc costs of the tree.

Note that both the AND and the OR case are here included, since in a solution graph OR nodes can have only one successor.

For example, let us consider the two solution oranhs in Fig. 4, where we assume unit arc costs. The cost of the first solution is 5 and the cost of the second one is 9, because the cost of the subtree rooted at T is considered twice.

In the next sections, we shall oive two aloorithms to find solution oranhs having minimal cost for additive AND/OR oranhs.

3) Top-down Search Aloorithm

The algorithm oiven in this section finds a solution oranh havino minimal cost, beoinnino with the start node s and using T- to oenerate the oranh. At everv staoe in the generation of the graph, a decision must be made about which node should have Its successors oenerated next. (Reneratino the successors of a node hv r Is called expanding a node.).

The aloorithm is an extension of the algorithm nronosed hv Nilsson for AND/OR trees and the two algorithms are practicallv identical for the narticular case of ANP/OR trees. The search method uses a set of arrows to ouide it from the start node down throuoh the AND/OR oranh searched thus far to that node to he exnanded next. Fach expanded node has an arrow pointing to one and only one of its successors based on evaluations of the successors, when a node is oenerated for the first time, its evaluation is an estimate, which must be available on separate "rounds, of the cost of a minimal cost solution graph starting at this node. The network of arrows is maintained bv updating the evaluations of nodes that are ancestral to those exnanded durino the process. The evaluation of a node gives the estimate of the cost of an optimal solution oranh havino that node as start node.

The aloorithm consists of the followinn steps:

(1) Begin with the start node- s,

(2) If s is not marked SOLVED, oo to (3); otherwise exit with the solution graph, This oranh is constructed by starting at s and using the final directions of the arrows to decide which successors of OR nodes should be included,

(3) Trace down the arrows from s to a node n and expand it. Suppose node n has successors $n_1 \ldots\ldots\ldots n_k$.

(4) Undate the evaluations and arrow directions of node n and all nodes ancestral to n, as follows:

Tf a successor n_1 of n has been oenerated before, it has already an evaluation $h(n_i)$. If a successor n_i is generated for the first time, its evaluation

$h(n_j)$ is the estimate of the cost of a minimal cost solution graph starting at node nj. If a successor is a terminal node, its evaluation is zero and it is marked SOLVED.

If node n is an OR node, its undated evaluation is

$$\hat{h}(n) = \min_i (\hat{h}(n_i) + c(n,n_i)).$$

The arrow is directed from n to the successor n_1 for which the minimum is achieved and n is marked SOLVED if and only if n_1 is marked SOLVED.

If node n is an AND node, its updated evaluation is

$$\hat{h}(n) = \sum_{i=1}^{k} (\hat{h}(n_i) + c(n,n_i)).$$

The arrow is directed from n to one of its successors which is not marked SOLVED, according to a given criterion. A reasonable heuristic is the one of directina the arrow to that successor having the largest evaluation. If all the successors are marked SOLVED, n is marked SOLVED.

The procedure of undating evaluations and arrow directions is then repeated for all the ancestors of node n by backing UP the granh to s.

(5) Go to (2)+

Note that during step (4), a node can he updated several times, because there can be several paths in the granh between a nair of nodes. However, the procedure will alwavs terminate, since in the graph there are no cycles by hypothesis.

An example of search with the given alonrithm is shown in Tin. 5. The nranh to he searched is shown in Fig. 5a. Arc costs are assumed to be unity and terminal nodes are marked. The numbers adjacent to every node are the estimate of the cost of a minimal cost solution araph starting at that node. Fig. 5b through 5g show the parts of the graph generated after each cycle of the algorithm. The numbers attached to each node are the updated evaluations and marked nodes are SOLVED nodes. Finally, Fig. 5h qives the solution graph found by the alnorithm. Actually, this solution granh has minimal cost, as we shall show below.

As in the case of the arrow algorithm of Nilsson2 or the algorithm of Hart, Nilsson and Raphael[9] for OR graphs, we can show that, if the estimate function h(n) is a lower bound on the cost of a minimal cost solution graph, then the algorithm is admissible, that is it will always find a minimal cost solution araph.

Let a tin node be a node of the aranh nenerated at a certain stage by the alnorithm which has not vet been expanded. Let h(n) he the cost of a minimal cost solution graph starting at node n. we then have:

Lemma 1. If $\hat{h}(n) \leq h(n)$ for all tin nodes, then at anv stane durinn the search process we have $\hat{h}(n) \leq h(n)$ for all nodes n in the graph. Moreover, if a node n is marked SOLVED we have $\hat{h}(n) = h(n)$ and an optimal solution aranh can he obtained hv tracing down the arrows from n.

Proof. We shall prove the lemma by induction on the stages of the algorithm. The lemma is trivially true at staae 0. Let us assume that it is true at a certain staae and let us nrove that it is true at the next stane, that is after the exnansion of a node (sav node n).

in sten *(A)* of the alnorithm, We update all the nodes which are ancestors of node n. Therefore, let us consider the subgraph G of the search aranh obtained un to this stage, which consists of all the ancestors of node n. Beinn the graph without cycles, an index can be attached to each node of G_f startina with $n^\circ = n$, in such a wav that all the naths from node n^1 to node n° contain only nodes n with $i < i$.

We shall nrove the lemma bv induction on the index i. The lemma is certainly true for node n. Tn fact, if anv of its successors n^ has been nenerated before, we have $\hat{h}(n_i) \leq h(n_i)$ bv induction and if the successor ni is generated for the first time, we have $h(n_j) < h(n_j)$ bv hypothesis. Therefore, bv computing h(n) according to sten (4) of the alnorithm, we have $\hat{h}(n) \leq h(n)$. Moreover, if node n is marked SOLVED and is an OP node, we know bv induction that for the successor n.| pointed to bv the arrow we have $h(n_i) = h(n_4)$ and that an optimal solution for n_4 can he obtained bv tracina down the arrows from n_j, An optimal solution aranh for node n will contain node n_j, since anv other solution nranh through another successor n_1 would have a not smaller cost. Tn fact, from

$$\hat{h}(n_i) \leq h(n_i)$$

we have

$$h(n_1) + c(n,n_1) = \hat{h}(n_1) + c(n,n_1) \leq$$

$$\leq \hat{h}(n_i) + c(n,n_i) \leq h(n_i) + c(n,n_i)$$

Thus, if node n is marked SOLVED and is on OR node, we have $\hat{h}(n) = h(n)$ and an optimal solution graph can he obtained bv tracing down the arrows from n. The same holds if node n is marked SOLVED and is an AND node.

Now, let us assume that the lemma is true for all the nodes n^j with $j < 1$; then the lemma is true for node n^i. This can be shown easily, by repeating for node n^i the above arguments for node n.

Q.E.D.

Now we can prove the following:

Theorem 1. If a solution graph exists, if $h(n) <. h(n)$ for all tip nodes n and if all arc costs are larger than some small positive amount 6, then the top-down algorithm is admissible.

Proof. We consider three cases:

Case 1. Termination without finding a solution graph. This case is impossible. In fact, termination can only occur at step (2) when the start node is solved. But this can only happen if a solution graph has been found.

Case 2. No termination. Let us examine the algorithm at a certain stage. If we trace down the arrows from node s, we obtain a path from node s to the node n to be expanded next. Let c(s,n) be the cost of this path, that is the sum of the costs of its arcs. For each are (n_j,n_j), belonging to this path, we have $h(n_i)>c(n_i,nj)+h(nj)$. Therefore, we have $h(s)> c(s,n)$. Let us assume now that, at a certain stage, we expand a node n for which $c(s,n)>h(s)$. This is not possible because it would imply $h(s) > h<s>$ contradicting Lemma 1. Therefore, the algorithm can only expand those nodes n for which $c(s,n) <_ h(s)$. The number of these nodes is finite because every node has a finite number of successors and the cost of every arc is greater than 6, hence the algorithm must terminate.

Case 3. Termination with a solution oraph having non minimal cost. When the alqorithm terminates, node s is marked SOLVED and we have $h(s) = h(s)$ by Lemma 1. Moreover, Lemma 1 states that an optimal solution graph can be obtained by tracing down the arrows from s. Therefore, the algorithm can only terminate with an optimal solution graph.

Q.E.D.

Hart, Nilsson and Raphael introduced an assumption, called the consistency assumption, for the estimate h(n), which allows a simplification of the algorithm for searching OR graphs. Analogously, we can give the consistency assumption for additive AND/OR graphs as follows. Let h(n) be the given estimate of the cost of a minimal cost solution graph starting at n. Then the consistency assumption is

if n is an OR node.

$h(n) <_ hfn^) + c(n,nj)$ for each successor n_1

if n is an AND node

$h(n) \leq \sum_1 (h(n_1) + c(n,n_i))$

With the consistency assumption, it is easy to see that the evaluation of each node can only increase at each stage of the algorithm. Therefore, we can modify the last statement of step (4) of the algorithm as follows. One step of the procedure of backinq up the graph of the ancestors of node n, consists of updatino all the parents of a certain node m. Assume that one of these parents m_i is an OR node. Since the evaluation of each node can only increase, the evaluation of node m_i will be changed only if node m_i has the arrow pointing to node m. Therefore, in the procedure of backinq up the qraph of the ancestors of node n, we can consider only those OR parents of a certain node m, which have the arrow pointing to m.Note that, even in this case, a node can be undated several times during step (4).

4) Bottom-up Search Algorithm.

This algorithm finds a solution graph having minimal cost, starting from the terminal nodes and expanding a node at each stage. In this case, expandinq a node means generating all the parent nodes of that node. The algorithm does not use any estimate, thus it can be considered as an extension of Dijkstra's algorithm for OR graphs[8]. An evaluation function is computed for each node and the node whose evaluation function is minimal is expanded. The evaluation function for a node n gives the cost of a minimal cost solution graph startina at node n, obtained so far.

The algorithm is based on the assumption that we have a finite number of terminal nodes and it consists of the following steps:

(1) Put every terminal node t_i on a list called OPEN and set $h(ti) = 0$ for every i

(2) Remove from OPEN that node whose h value is smallest and put it on a list called CLOSED. (Resolve ties arbitrarily). Call this node n.

(3) If n is the start node, exit with the solution graph obtained by tracing back through the pointers, otherwise continue.

(4) Expand node n, generating all of its parents. For each parent n_i do the following:

If n_i. is an OR node, we can have three cases

a) n_i is neither on OPEN nor on CLOSED. Associate with n_i the value of the evaluation function

$\tilde{h}(n_1) \cdot \tilde{h}(n) + c(ni,n)$

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

b) n_i is on OPEN. Let h(ni) be the evaluation associated with n_i. Associate with it a new value of the evaluation function given by

$$\hat{h}'(n_i) = \min (\hat{h}(n_i), \hat{h}(n) + c(n_i, n))$$

If $h(n_i)$ has been changed, redirect the pointer from n_i to n.

c) n_i is On CLOSED. Continue

If n_i is an AND node, we can have two cases

a) There is some successor of n_i which is not on CLOSED. Continue.

b) All of the successors of n_i are on CLOSED. Associate with node n_i the value of the evaluation function

$$\hat{h}(n_i) = \sum_k (\hat{h}(n_k) + c(n_i, n_k)),$$

where the nodes n_k are the successors of n_i. Direct pointers from n_i to all of its successors and put n_i on OPEN.

(5) Go to (2).

This algorithm is closely related to the dynamic programming method. In fact, the latter can be seen as a bottom-up breadth--first search algorithm, which expands all the nodes according to a fixed ordering. Our algorithm, on the contrary, expands the nodes according to their cost, thus expanding, in general, fewer nodes.

An example of search with the given algorithm is shown in Fig. 6. The graph to be searched Is given in Fig. 5a. The estimates attached to its nodes are not used by this algorithm. Figs. 6a through 6j show the parts of the graph generated after each cycle of the algorithm. Marked nodes are the closed nodes and the other nodes are the open nodes. Arc costs are assumed to be unity and the numbers adjacent to the nodes are the values of h. Nodes without an adjacent number are AND nodes which have some successor not yet closed. Fig. 6k gives the solution graph.

Note that the graph generated by this algorithm has a simpler structure than the one generated by the top-down algorithm. In fact, at every stage of the algorithm, an OR node has only one pointer to one if its successors whereas, at every stage of the top-down algorithm, the complete structure of the graph has to be retained.

We can show that the bottom-up algorithm is admissible, that is, it will always terminate in an optimal solution graph, whenever a solution graph exists.

Lemma 2. If all arc costs are positive, then, when a node n is closed by the algorithm, we have $\hat{h}(n) = h(n)$ and the optimal solution graph can tracing down the pointers from n. Moreover, all the nodes m with h(m) < h(n) have been closed before node n.

Proof. The lemma will be proved by induction on the stages of the algorithm. The lemma is trivially true at stage 0. Let us assume it is true at a certain stage and let us prove that it will still be true after the next stage.

Let n be the open node whose h value is smallest at this stage. This node is closed by the algorithm. Let us assume that a node m exists with h(m) <h(n), which has not yet been closed by the algorithm. Let G be an optimal solution graph for node m. Certainly, there is an open node of G, such that all of its successors in G are closed. It is possible to see that $\hat{h}(r) = h(r)$. In fact, if r is an AND node all its successors are closed and by the induction hypothesis their h value is the cost of a minimal cost solution graph, hence $\hat{h}(r) = h(r)$.

If r is an OR node, we have

$$\hat{h}(r) = \min_i (\hat{h}(r_i) + c(r, r_i))$$

where the nodes r_i are the successors of r which have been closed so far. But we assumed that one of these successors belongs to an optimal solution graph for r, therefore $\hat{h}(r) = h(r)$.

Moreover we have

$$h(m) \geq h(r)$$

since the optimal solution graph for r is a subgraph of G, and

$$\hat{h}(n) \leq \hat{h}(r)$$

since n is the open node whose h value is smallest. Therefore we have

$$\hat{h}(n) \leq \hat{h}(r) = h(r) \leq h(m)$$

which contradicts our hypothesis that node m has not yet been closed and we have shown that all the nodes m with h(m)<h{n) have already been closed.

Now, we can show that $\hat{h}(n) = h(n)$. If n is a terminal node, this is obviously true. If n is an OR node, we have shown above that, for any successor ni of n which is not closed, we have $h(ni) \geq h(n)$. Therefore, an optimal solution graph must contain a closed successor, because any solution

graph through one successor which is not closed would have a larger cost. The pointer is directed from n to the closed successor n for which $(h(n_j) + c(n,njl)$ Is minimal and, by the induction hypothesis, $h(n_j)=h(n_j)$. Therefore, we have h(n) = h(n) and an optimal solution graph for n can be obtained by tracing down the pointers.

If n is an AND node, all of its successors are already closed. Since the lemma is true for the successors by the Induction hypothesis, it is also true for n.

Q.E.D,

Finally, we can prove:

Theorem 2. If a solution graph exists and if all arc costs are positive, then the bottom-up search algorithm is admissible.

Proof. We prove the theorem by assuming the contrary. There are three cases to consider:

Case 1. Termination without finding a solution graph. This case is impossible because the algorithm can only terminate at step (3) with a solution graph.

Case 2. No termination. Assume that at a certain Btage the algorithm closes node n with h(n) > h(s). This is impossible, because, according to Lemma 2, node s must have been closed before and when s is closed the algorithm terminates. Therefore, the algorithm can only close those nodes n with h(n) <_ h(s). But the number of these nodes is finite, since the arc costs are positive, and the algorithm must terminate.

Case 3. Termination with a solution graph having nonmlnlmal cost. This is not possible, because it would contradict Lemma 2.

Q.E.D.

Refefences.

1 Nilsson, N.J., Problem-Solving Methods in Artificial Intelligence, Mc Graw-Hill, New 'fork, 1971.

2 Nilsson, N.J., Searching Problem-Solving and Game-Playing Trees for Minimal Cost Solutions, Proc. IFIF Congress 1968, H, pp. 125-130.

3 Chang, C.L, and Slagle, J.R., An Admissible and Optimal Algorithm for Searching AND/OR Graphs, Artificial Intelligence, Vol. 2, pp. 117- 128 (1971).

4 Martelli, A. and Montanari, u., Dynamic Programming via AND/OR graphs, in preparation.

5 Knuth, D.E., optimum Binary Search Trees, Acta Informatlca, Vol. 1, pp. 14-25 (1971).

6 Martelli, A,, Edge Detection Using Heuristic Search Methods, Computer Graphics and_ Image Processing,Vol. 1, N. 2, pp. 169-182, August 1972.

7 Kaufmann, A. and Cruon, R., Dynamic Programming, Academic Press, 1967.

8 Dijkstra, E., A Note on Two Problems In Connection with Graphs, Numerlsche Mathematlk, Vol. 1, pp. 269-271 (1959).

9 Hart, P., Nilsson, N.J, and Raphael, B., A formal Basis for the Heuristic Determination of Minimum Cost Paths, I.E.E.E. Trans. SCC 4, N.2, pp.100-107, July 1968.

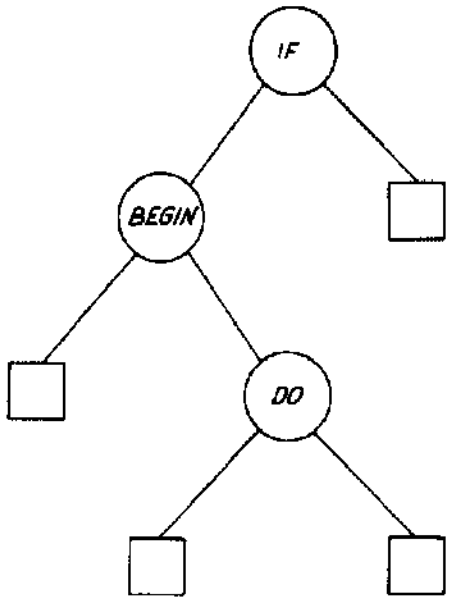

Fig. 1 - A binary search tree.

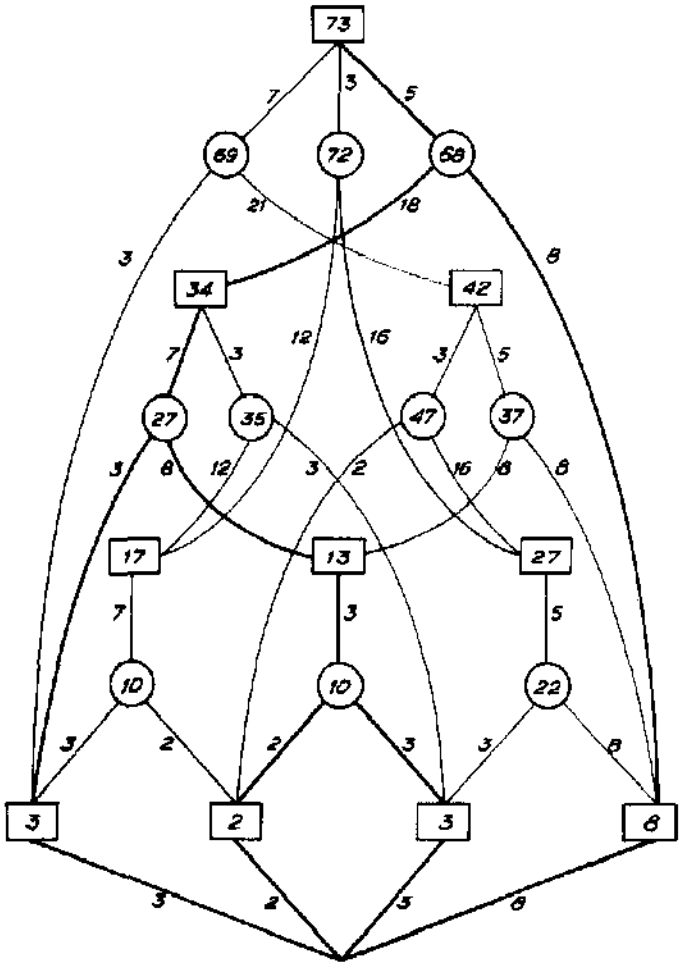

Fig. 2 - AND/OR graph for the optional search tree problem.

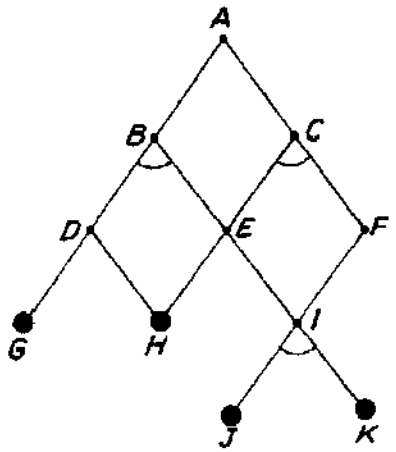

Fig. 3 - An AND/OR graph without cycles

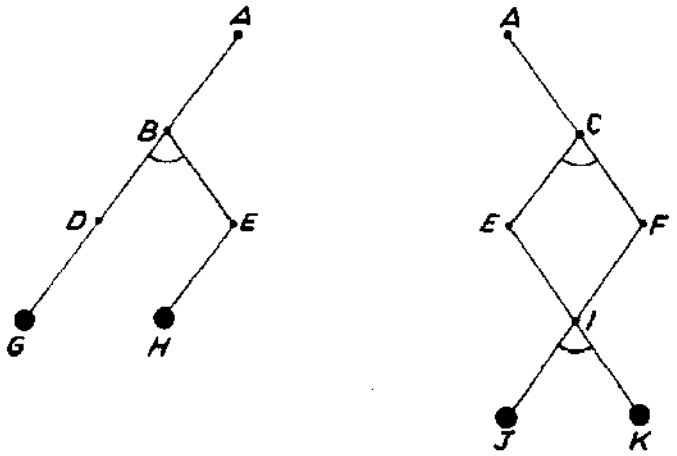

Fig. 4 - Two solution graphs of the AND/OR graph in Fig. 3.

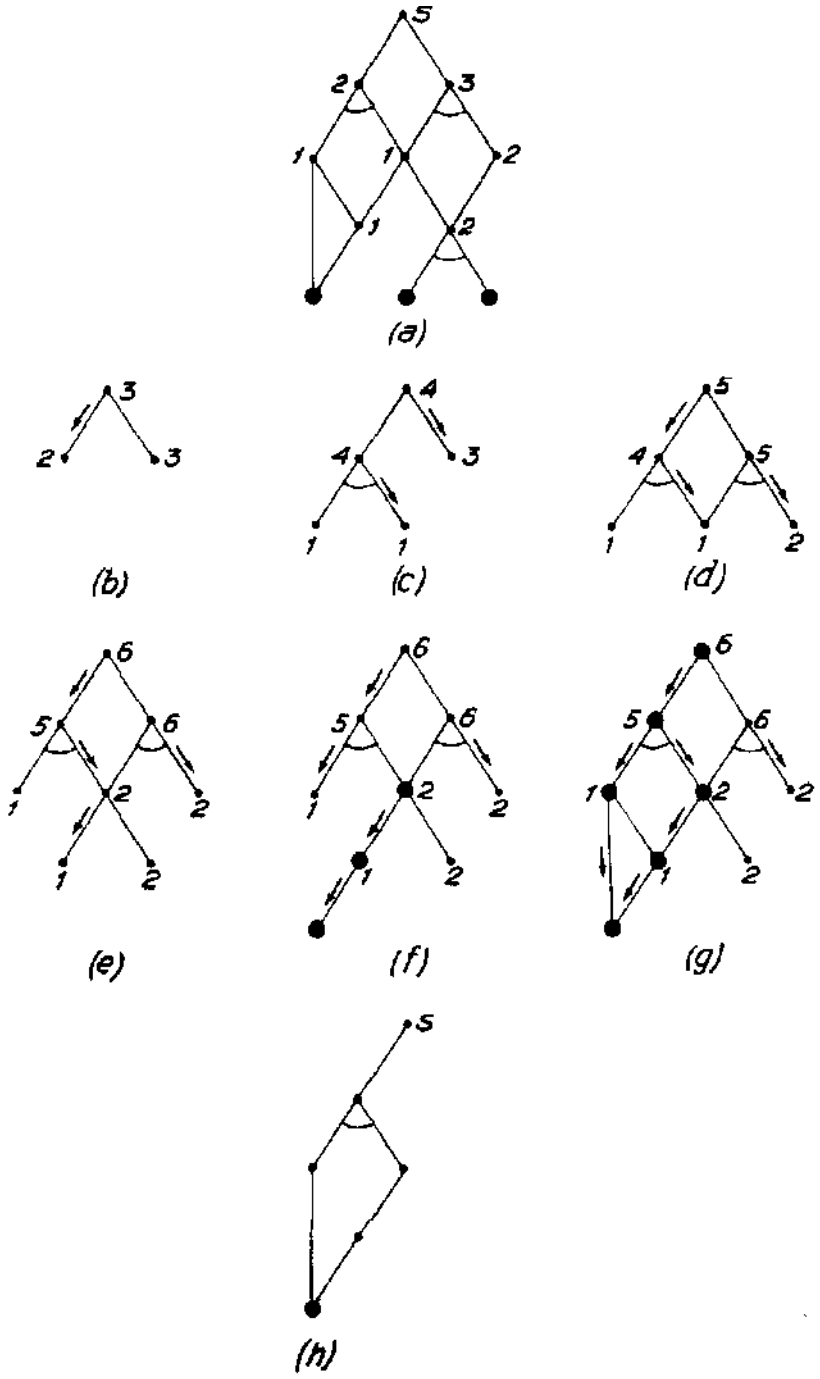

Fig. 5 - An example of search with the top-down algorithm.

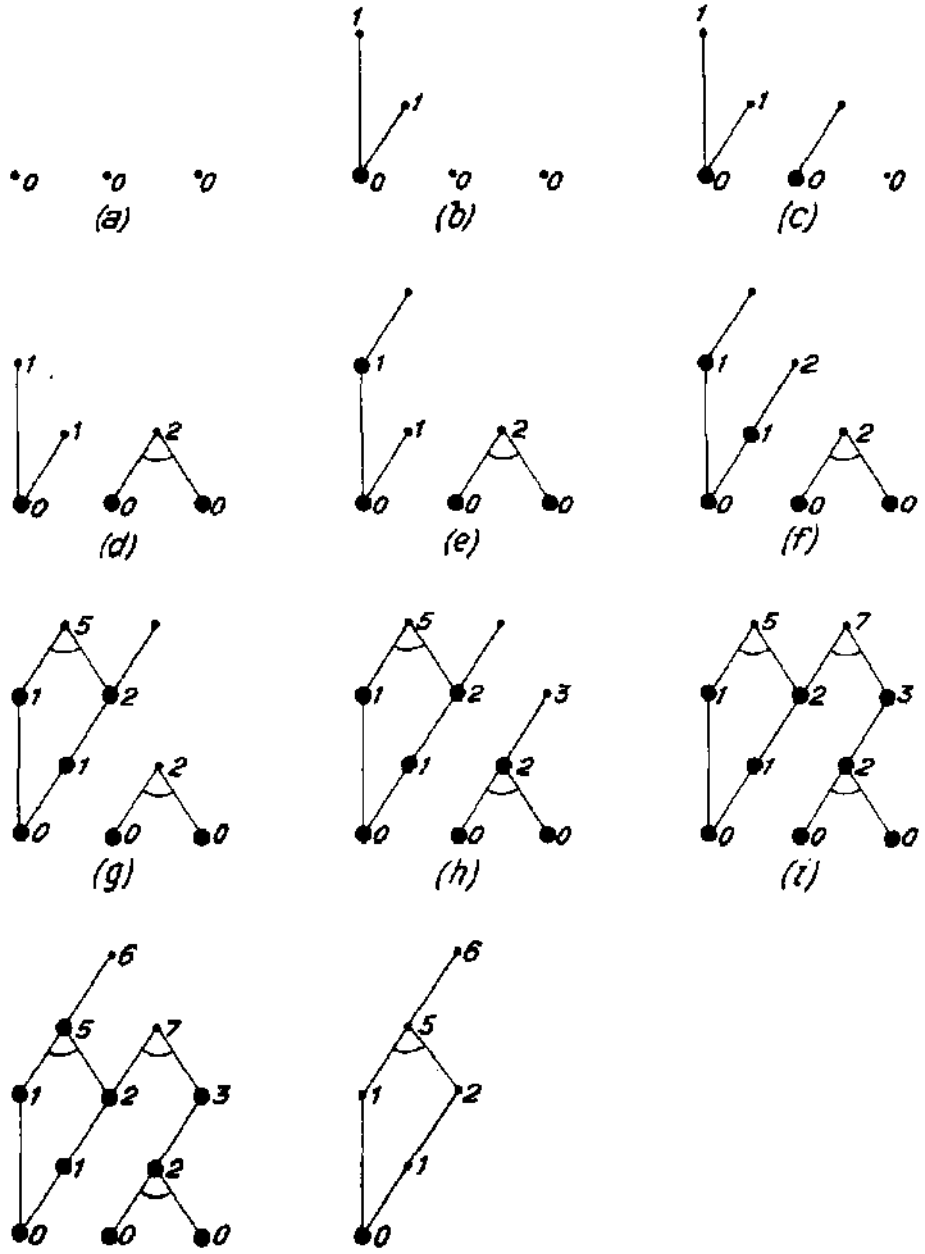

Fig. 6 - An example of search with the bottom-up algorithm.