A Generalization of the AO* Algorithm

by

Vipin Kumar*, Dana S. Nau**, and Laveen N. Kanal**

ABSTRACT

This paper presents a general search procedure called GAO*. GAO* is a generalization of AO* which finds optimal solution trees in acyclic AND/OR graphs having monotone cost functions. Since monotone cost functions are very general, GAO* is applicable to a very large number of problems. For example, many game tree search procedures (e.g., B*, SSS*) are variations of GAO*. The proof of correctness of GAO* is quite simple, which simplifies the correctness proof of AO*. This work is important in the context of authors' previous work on a unified approach to search procedures.

Topic: Automated Reasoning

Subtopic: Search

Key words: AND/OR graph search, branch-and-bound, game tree search, monotone cost functions.

Paper length: approximately 2200 words

---

* Artificial Intelligence Laboratory
Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712
Arpanet: kumar@ut-sally.arpa
Tel: (512) 471-7316

** Machine Intelligence and Pattern Analysis Laboratory
Department of Computer Science
University of Maryland
College Park, MD 20742

This work was supported in part by by Army Research Office grant #DAAG29-84-K-0060 to the Artificial Intelligence Laboratory at the University of Texas at Austin, in part by a NSF Presidential Young Investigator Award to Dana S. Nau at the University of Maryland, and in part by NSF grants to the Machine Intelligence and Pattern Analysis Laboratory at the University of Maryland.

A Generalization of the AO* Algorithm

1. INTRODUCTION

Many Artificial Intelligence problems can be formulated as: "Given an AND/OR graph with certain cost functions associated with the arcs, find a least-cost solution tree". This paper presents a general heuristic top-down procedure, GAO*, for solving such problems when the AND/OR graph is acyclic and the cost functions associated with the arcs are monotone. GAO* is a generalization of the AO* algorithm [8] [10] for searching AND/OR graphs,[1] and is a kind of Branch-and-Bound procedure. Since monotone cost functions are very general, GAO* is applicable to a very large number of problems. For example, GAO* can be used to find the minimax value of a game tree. Furthermore, many existing game tree search procedures (e.g., B* [1] and SSS* [12]) can be considered as variations of GAO*. The proof of correctness of GAO* is quite simple, which simplifies the correctness proof of AO*.

2. AND/OR Graphs

Following the terminology in [10], [8], we define AND/OR graphs as hypergraphs. Each node of an AND/OR graph represents a problem, and the root of G (denoted by root(G)) represents the original problem to be solved. If a problem n can be solved by solving a set of subproblems $n_1,...,n_k$, this is depicted in the hypergraph by a *hyperarc* or k-*connector* $p: n \rightarrow n_1,...,n_k$ directed from the node n to the child nodes $n_1,...,n_k$. A node can have more than one hyperarc directed from it.

An AND/OR graph G is *acyclic* if no node of G is an ancestor of itself. Every acyclic 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).

---

[1] AO* finds a least-cost solution tree of an acyclic AND/OR graph when the cost functions associated with the arcs are additive (which is a special case of monotone functions).

Given an AND/OR graph G representing a problem, each solution to the problem will be represented by a *solution tree* for G, which is a subtree T of unfold(G) having the following properties:

(i) root(G) = root(unfold(G)).

(ii) if a nonterminal node n of unfold(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 unfold(G). By a *solution tree rooted at n* we mean a solution tree for the subgraph of G rooted at n.

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 $p: n \rightarrow n_1,...,n_k$ we associate a k-ary cost function $t_p(r_1,...,r_k)$ which denotes the cost of solving n if n is solved by solving $n_1,...,n_k$ at costs $r_1,...,r_k$, respectively.

For a solution tree T, we define its cost f(T) recursively as follows (an example appears in Fig. 1):

2.1a if T consists only of a single node n = root(T), then f(T) = c(n).

2.1b Otherwise, n = root(T) has children $n_1,...,n_k$ such that $p: n \rightarrow n_1,...,n_k$ is a connector. Let $T_1,...,T_k$ be the subtrees of T rooted at $n_1,...,n_k$. Then $f(T) = t_p(f(T_1),...,f(T_k))$.

Let $c^*(n)$ be the minimum of the costs of the solution trees rooted at n. Then $c^*(root(G))$ is the cost of an optimum solution tree for G. The following theorem gives a recursive formula for $c^*(n)$.

Theorem 2.1: If G is an acyclic AND/OR graph whose cost functions $t_p(.,...,.)$ are monotonically nondecreasing in each variable, then for every node n of G 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
$$c^*(n) = \min\{t_p(c^*(n_1),...,c^*(n_k)) \mid p: n \rightarrow n_1,...,n_k \text{ is a hyperarc directed from } n\}.$$

Proof: See [5].

2.2 Maximization problems

In many problem domains, f(T) denotes the merit of the solution tree T, and a solution tree of largest merit is desired. In such cases, c(n) denotes the merit of a terminal node n of G.

The functions $t_p(.,...,.)$ and f are defined exactly as before, but $c^*(n)$ is the maximum of the merits of the solution trees rooted at n. In this case, Theorem 2.1 can be restated with "min" replaced by "max" in its second condition.

2.3 Versatility of Monotone Functions

The monotone functions are a wide class of functions. A number of useful cost (or merit) functions are monotone. Examples are given below.

(1) If $t_p(x_1,...,x_k) = x_1+...+x_k$, then f(T) is the total number of terminal nodes in T.

(2) If $t_p(x_1,...,x_k) = 1 + \max\{x_1,...,x_k\}$, then f(T) is the depth of T.

(3) Let $t_p(x_1,...,x_k) = c_p + x_1 + ... + x_k$, where c_p is the cost of applying the reduction operator p. Then f(T) is the sum of the costs of solving the terminal problems of T and applying the problem reduction operators. This is the cost function used by AO* in [10] [8].

(4) Let $t_p(x_1,...,x_k) = \min\{x_1,...,x_k\}$ in a maximization problem (as discussed in Section 2.2). Then c*(root(G)) is the minimax value of root(G) if G is viewed as a game tree (for a proof see [12], [6]). Thus a procedure for searching AND/OR graphs with monotone cost functions can be used to find the minimax value of a game tree.[2]

3. A General Heuristic Top-down Search procedure

This section presents the details of GAO*. GAO* assumes the existence of a heuristic "lower bound" function b defined over the nodes n of G such that $b(n) \leq c^*(n)$; i.e., b(n) is a lower bound on the cost of an optimal solution tree rooted at n. This function is used by GAO* to speed up the search. We further assume that b is "heuristically consistent"; i.e., for each connector $p: n \rightarrow n_1,...,n_k$, $b(n) \leq t_p(b(n_1),...,b(n_k))$. This property implies that the lower bound of a node computed by looking at its successors will never be worse than the lower bound associated with the node.

[2] For example, Pearl [11] uses a variation of AO* to search game trees.

Procedure GAO*

(1) The initial graph consists only of the node root(G).

(2) Repeat the following steps until root(G) is labeled SOLVED; then stop.

(2.1) Select any tip node n of the solution tree obtained by tracing down the marked connectors from root(G).

(2.2) Expand n by generating all of its successors. For each n_j, set $b^*(n_j) = b(n_j)$.

(2.3) Create a set of nodes S containing only n.

(2.4) Repeat the following steps until S is empty.

(2.4.1) Remove from S a node n such that no other node in S is a successor of it.

(2.4.2) Update $b^*(n)$ as follows: for each connector p: $n \rightarrow n_1,...,n_k$, compute $t_p(b^*(n_1),...,b^*(n_k))$. Set $b^*(n)$ to minimum of these values, and mark the connector through which the minimum is achieved. If this n is a terminal node or if all of the children of n in the marked connector are labeled SOLVED, then label n SOLVED.

(2.4.3) If n has been marked SOLVED or if $b^*(n)$ has increased, add to S all parents m of n such that there is a marked connector from m to n.

Let G' be the graph generated by GAO*. For every node of G', GAO* maintains a value $b^*(n)$ which is an estimate (lower bound) of $c^*(n)$. G' initially consists of just root(G). In each cycle, GAO* selects a tip node of G' and expands it. When a node n is generated, $b^*(n)$ is initialized to b(n), and the b^*-values of the parents of n are appropriately revised.

The Correctness of GAO*

The correctness of GAO* follows from the following theorem, because GAO* terminates only when the root node is labeled SOLVED.

Theorem 3.1: If a node n is labeled SOLVED by GAO*, then $b^*(n) = c^*(n)$, and a least-cost solution tree T rooted at n (i.e., a tree T rooted at n such that $f(T) = c^*(n)$) can be found by following the marked connectors from n.

Proof: By induction on the height of n in G'.[3]

Base case: the height of n is 0; i.e., n is a tip node of G'.
If n is labeled SOLVED then n must be a terminal node of G; hence $b^*(n) = b(n) = c^*(n)$.
There are no marked connectors going out of n, and the least-cost solution tree rooted at n consists of n itself.

---

[3] Here, the height of n is the length (i.e., the number of arcs) in the longest path from n to a tip node of G'.

Induction step: suppose the theorem holds for all nodes of height h or less, and let n be any node of height h+1.

If n is labeled SOLVED, then there must be a connector $p: n \rightarrow n_1,\ldots,n_k$ such that $n_1,\ldots,n_k$ are labeled SOLVED. Since n has height h+1, the heights of nodes $n_1,\ldots,n_k$ must each be h or less. Thus from the induction assumption, $c^*(n_i) = b^*(n_i)$ for $1 \leq i \leq k$. Thus

$$\begin{aligned} b^*(n) &= t_p(b^*(n_1),\ldots,b^*(n_k)) \quad \text{(from step (2.4.2))} \\ &= t_p(c^*(n_1),\ldots,c^*(n_k)) \quad \text{(from the induction assumption and since n is SOLVED)} \\ &\geq c^*(n) \quad \text{(from Theorem 2.1).} \end{aligned}$$

But from Theorem A.1, for all nodes n of G', $b^*(n) \leq c^*(n)$. Therefore,

$$b^*(n) = c^*(n). \tag{3.1}$$

Let T be the solution tree constructed by following the marked connectors from n. T must have subtrees $T_1,\ldots,T_k$ rooted at $n_1,\ldots,n_k$ such that $T_1,\ldots,T_k$ are formed by following marked connectors from $n_1,\ldots,n_k$. From the induction assumption, $f(Ti) = c^*(n_i)$ for each i. Thus

$$\begin{aligned} f(T) &= t_p(f(T_1),\ldots,f(T_k)) \quad \text{(from the definition of f)} \\ &= t_p(c^*(n_1),\ldots,c^*(n_k) \quad \text{(from the induction assumption)} \\ &= c^*(n). \quad \text{(from eq. (3.1))} \end{aligned}$$

AO* as a special case of GAO*

If the cost functions are of the form $t_p(x_1,\ldots,x_k) = c_p + x_1 + \ldots + x_k$ (where c_p is the cost associated with the connector p), then GAO* becomes identical HS [8] (a version of AO* [10]). The heuristic consistency property of the lower-bound function b is the same as the consistency property in [8] and the "monotone restriction" in [10].[4] As discussed in [10] in the context of AO*, the heuristic consistency property of b is not crucial for the correctness of GAO*. It merely reduces the work done in Step 2.4 of GAO*.

GAO* as Branch-and-Bound

GAO* also has a natural interpretation as the kind of Branch-and-Bound (B&B) algorithm described in [9]. G' can be viewed to represent a set of "partial solution trees" (i.e., partially explored solution trees) in exactly the same way that G represents a set of solution trees. Each partial solution tree T' of G' represents the set of all solution trees of G which are extensions of

---

[4] Note that the monotone restriction on the lower bound function in Nilsson has no relation with the monotonicity of the cost functions as defined in this paper.

T'. GAO* has an implicit lower bound $f_b(T')$ on the cost of solution trees represented by T'.[5] In Step 2.1, by following marked connectors, GAO* selects a partial solution tree T' of G having the smallest lower bound. Expanding a node of T' is essentially equivalent to splitting the set of solution trees represented by T'.

After root(G') is labeled SOLVED, the partial tree T' found by the following marked connectors has all of its tip nodes as terminal nodes in G; i.e., T' is a complete solution tree. Therefore, $f_b(T') = f(T')$. At this point, other partial trees have higher lower bounds than the cost of T'; hence T' is guaranteed to be a least-cost solution tree of G. Thus GAO* terminates.

Variations of GAO*

From Theorems 3.1 and A.1, it is clear that GAO* would still work properly even if nodes were chosen in a different manner from what is specified in Step 2.1. Steps 2.2, 2.3 & 2.4 ensure the validity of Theorems 3.1 & A.1. Hence *any* partial solution tree T' of G' can be selected in Step 2.1, and any tip node node n of T' can be expanded in Step 2.2.[6] By making different choices in Step 2.1, many variations of GAO* can be produced; some of them are equivalent to some well known procedures. For example, SSS* [12] can be viewed as a variation of GAO*. SSS* assumes that the bound function b gives no information except on terminal nodes,[7] which makes it beneficial to use a different criterion in Step 2.1 for selecting a most promising partial solution tree. B*, which uses heuristic information to search game trees, can also be looked at as a variation of GAO*. Being a game tree search procedure, B* tries to find only the immediate successor of the root node in a largest merit solution tree. This lets it use a somewhat different termination criterion in Step 2 and two different node selection policies ("prove-best"

[5] If defined explicitly, $f_b(T')$ would (analogously to f) be $f_b(T') = t_p(f_b(T'_1),...,f_b(T'_k))$.

[6] The rationale behind the current choice in Step 2.1 of GAO* is that if the heuristic function is good, GAO* can find an optimum solution tree very quickly.

[7] In maximization problems such as game tree searching, b is taken to be an upper bound. SSS* assumes that $b(n) = +\infty$ if n is nonterminal.

and "disprove-rest") in Step 2.1. The prove-best selection policy is same as the one used by GAO*. For details, the reader is refered to [7].

4. Concluding Remarks

The development of GAO* was inspired by comparing AO* and several game tree search procedures with B&B procedures [9] [6] [5]. There has been much confusion about the relationships among various search procedures (examples are given in [5] [2]), and the close relationships between GAO* and search procedures such as AO*, SSS*, alpha-beta, and B* clarifies the nature of these procedures.

Since monotone cost functions are very general, GAO* is applicable to a large number of problems. In addition, the simple correctness proof of GAO* provides an easy way to verify the accuracy of these related search procedures.

Our work on a unified approach to search procedures has resulted in the synthesis of new algorithms (e.g., generalizations, variations, and parallel implementations of various search procedures [6] [4] [3]).

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

[2] Kumar, Vipin, The Composite Decision Process: A Unifying Formulation for Heuristic Search, Dynamic Programming, and Branch and BOund Procedures, *AAAI-83*, Washington, DC, pp. 220-224, Aug. 1983.

[3] Kumar, Vipin, A General Bottom-Up Procedure for Searching AND/OR Graphs, *AAAI-84*, Austin, TX, 1984.

[4] Kumar, Vipin and Kanal, Laveen, Parallel Branch and Bound Formulations for AND/OR Tree Search, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Nov. 1984.

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

[6] Kumar, V. and Kanal, L., A General Branch and Bound Formulation for Understanding and Synthesizing And/Or Tree Search Procedures,, *Artificial Intelligence*, 1983.

[7] Kumar, V., Nau, D. S., and Kanal, L. N., A General Paradigm for AND/OR Graph and Game Tree Search, in preparation, 1985.

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

[9] Nau, D. S., Kumar, V., and Kanal, L. N., General Branch and Bound, and Its Relation to A* and AO*, *Artificial Intelligence* **23**, pp. 29-58, 1984. (Also available as Tech. Report TR-1170, Computer Science Dept., Univ. of Maryland, 1982.)

[10] Nilsson, N. J., *Principles of Artificial Intelligence,* Tioga, Palo Alto, 1980.

[11] Pearl, Judea, *Heuristics: Intelligent Search Strategies for Computer Problem Solving,* Addison-Wesley, Reading, Mass., 1984.

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

Appendix

Theorem A.1: In GAO*, for all nodes n of G', $b^*(n) \leq c^*(n)$.

Proof: By induction on the height of n in G'.

Base case: the height of n is 0; i.e., n is tip node of G'.

Then $b^*(n) = b(n) \leq c^*(n)$ (by definition of b).

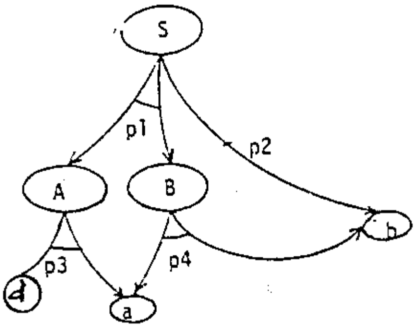
Induction step: suppose the theorem holds for all nodes of height h or less, and let n be any node of height h+1.

Step 2.4 and the heuristic consistency property ensure that every time a node is expanded (causing a change in G'), the following equation holds:

(A.1) $b^*(n) = \min\{t_p(b^*(n_1),\ldots,b^*(n_k)) \mid p: n \rightarrow n_1,\ldots,n_k \text{ is a connector in } G'\}$

From Theorem 2.1, (A.2) $c^*(n) = \min\{t_p(c^*(n_1),\ldots,c^*(n_k)) \mid p: n \rightarrow n_1,\ldots,n_k \text{ is a connector in } G'\}$

But from the induction assumption, $b^*(n_i) \leq c^*(n_i)$ for all i. Thus since t_p is monotonic, it follows that from A.1 and A.2 that $b^*(n) \leq c^*(n)$.



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.
c(d)= 4.

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

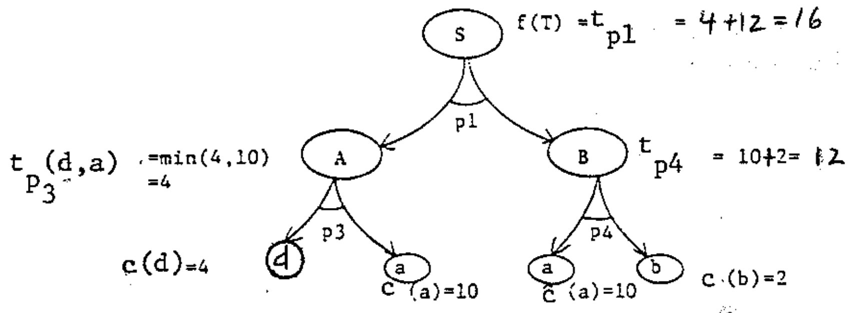


Fig. 1(b). Computation of f(T) of a solution tree T of G.