Optimal Satisficing Tree Searches

Dan Geiger and **Jeffrey A. Barnett**
Northrop Research and Technology Center
One Research Park
Palos Verdes, CA 90274

Abstract

We provide an algorithm that finds optimal search strategies for AND trees and OR trees. Our model includes three outcomes when a node is explored: (1) finding a solution, (2) not finding a solution and realizing that there are no solutions beneath the current node (pruning), and (3) not finding a solution but not pruning the nodes below. The expected cost of examining a node and the probabilities of the three outcomes are given. Based on this input, the algorithm generates an order that minimizes the expected search cost.

Introduction

Search for satisfactory solutions rather than optimal ones is common in many reasoning tasks. For example, a theorem prover may search for an acceptable proof although that proof is not necessarily the shortest possible. Similarly, planning the way home from a friend's house does not require us to look for the shortest path, any reasonable path suffices.

Simon and Kadane (1975) examine satisficing search using a simple gold-digging example: An unknown number of treasure chests are randomly buried at some of n sites, but neither the sites nor the depth of burial are known with certainty. At each site, a sequence of one-foot slices can be excavated, and a treasure may be disclosed by the removal of any one of these slices. The probability that a treasure lies just below each slice is known as is the cost of excavating that slice. Which search strategy minimizes the expected cost to find a treasure?

If slices can be excavated in arbitrary order, the optimal search strategy is to excavate slices in decreasing order of their benefit-to-cost ratios. However, there is a constraint: a slice can be excavated only after all slices above it are excavated. Consequently, a greedy approach—selecting the currently most promising slice—is not adequate. One should prefer to excavate a slice with a low benefit-to-cost ratio if a sufficiently promising slice lies under it. Simon and Kadane provide a method to find excavation sequences with the least expected cost to find a treasure.

This article defines a more detailed model of search where excavating a slice may prune search beneath that slice. Simon and Kadane's characterization of optimal excavation sequences is shown valid in our model and a variant of Garey's (1973) algorithm is developed to find these sequences for trees.

An example of an OR graph is given where the optimal strategy must be constructed dynamically. The example stands in sharp contrast to Simon and Kadane's model where optimal search strategies are determined before search starts. Finally, optimal searches of AND-OR trees are shown to require dynamic strategies too.

Search of OR Trees: Preliminaries

We extend Simon and Kadane's search model to allow three rather than two outcomes for excavating a slice: (1) a treasure is found, (2) a treasure is not found and it is realized that there are no treasures beneath the current slice (pruning), and (3) a treasure is not found but one can still be found below. In the later case, several alternatives are revealed for further digging. The respective probabilities of the three outcomes for the slice s are $p^+(s)$, $p^-(s)$, and $p^0(s)$. These outcomes are assumed to be mutually-exclusive and exhaustive and, hence, $p^+(s) + p^-(s) + p^0(s) = 1$. Simon and Kadane exclude the second outcome because they do not model pruning.

Metaphorically, if a treasure is not found at a slice, then either (1) some "doors" to new slices open—a situation that corresponds to exposing the immediate children of a node in a search tree or (2) no doors open—a situation that corresponds to either reaching a leaf node or pruning deeper search through that node. We assume that each slice is part of a single site and that each slice can be reached by only one path from the surface. Consequently, a site is a metaphor for

In *Proceedings of Ninth National Conference on Artificial Intelligence* (AAAI–91) 441–445

a tree and multiple sites for a forest.[1] More general searches are discussed later.

A sequence of slices $b = s_1 \dots s_r$ is a (search) *strategy* when (1) the s_i are distinct slices and (2) all slices above each s_i are in b and all precede s_i.

Define $\phi^+(s) = p^+(s)/c(s)$ as a benefit-to-cost ratio with the assumption that $p^+(s) \neq 1$ and $c(s) \neq 0$. These assumptions entail, respectively, that no slice contains a treasure with certainty and that no slice is excavated for free. Assume, also, that the probabilities and cost for one slice are independent of the outcome of excavating other slices.

The cost of a strategy $b = s_1 \dots s_r$, denoted $c(b)$, is computed by

$$c(b) = \sum_{i=1}^{r} \beta(s_i|s_1 \dots s_{i-1}) \cdot c(s_i), \quad (1)$$

where $\beta(x|s_1 \dots s_z)$ stands for the probability that slice x is excavated given the strategy starts with $s_1 \dots s_z$. Similarly, the probability that a strategy b unearths a treasure, denoted $p^+(b)$, is computed by

$$p^+(b) = \sum_{i=1}^{r} \beta(s_i|s_1 \dots s_{i-1}) \cdot p^+(s_i).$$

Formulas for β are developed below. For each strategy b, define $\phi^+(b) = p^+(b)/c(b)$ and $q^+(b) = 1 - p^+(b)$.

The problem is to find a strategy having the least expected cost, i.e., to find a strategy b^o such that $c(b^o) \leq c(b)$ for every strategy b.

In Simon and Kadane's search model, a slice s_i in a strategy $b = s_1 \dots s_r$ is excavated if and only if none of $s_1 \dots s_{i-1}$ contain a treasure. In this case, the expected cost of b is derived from Eq. (1) by substituting

$$\beta(s_i|s_1 \dots s_{i-1}) = \prod_{j=1}^{i-1} q^+(s_j). \quad (2)$$

This equation states that the probability that s_i is excavated equals the probability that no treasure is found in slices $s_1 \dots s_{i-1}$.

In our search model where pruning is permitted, the expected cost of a strategy is still given by Eq. (1), however, the expression for β is more complex. A slice is excavated if and only if every slice above it opens its doors (i.e., the path to that slice is unearthed) and no treasure is found by prior excavation.

[1]The word-pairs, site/tree, slice/node, and excavation/search are used interchangeably throughout this article to emphasize the analogy between the gold-digging example and tree searches.

When there is only one site, a tree T, then β is defined by

$$\beta(s_i|s_1 \dots s_{i-1}) = \prod_{j=1}^{n} p^0(a_j) \prod_{\substack{d \in K(a_j) \\ d \neq a_{j+1}}} Q^+(d) \quad (3)$$

$$Q^+(d) = p^-(d) + p^0(d) \prod_{x \in K(d)} Q^+(x),$$

where $K(d)$ is the set of children of node d that are in $\{s_1 \dots s_{i-1}\}$ and $a_1 \dots a_n$ is the path from the root of T to $s_i = a_{n+1}$. The formula for $Q^+(d)$ computes the probability that a subtree rooted at node d does not contain a treasure; when d is a leaf node, then $Q^+(d) = p^-(d) + p^0(d) = q^+(d)$.

When there are several sites, i.e., a forest, the expression for $\beta(s_i|s_1 \dots s_{i-1})$ in Eq. (3) is multiplied by $Q^+(r)$ for each root node, r, in $s_1 \dots s_{i-1}$ besides the root of T. The original formula calculates the probability that a path to s_i is unearthed and that no treasure is found in T prior to excavating s_i. The additional factors account for the assertion that no treasure is found at the other sites either.

Thus, the calculation of β in Eq. (3) depends on the topology of the sites as just described and on s_i itself because its ancestors, $a_1 \dots a_n$, are distinguished in the formula. Eq. (2) depends on neither.

Search of OR Trees: An Algorithm

A brute force approach for choosing the best excavation strategy computes the cost of each strategy and chooses the least expensive. Fortunately, when two strategies are identical except that two adjacent slices are switched, one can choose between the two strategies without computing their expected costs; merely compare the benefit-to-cost ratios, ϕ^+, of the slices that are switched, and choose the strategy where the slice with the highest ratio is excavated first. This local property facilitates a polynomial-time algorithm to find an optimal strategy. The next theorem spells out this property.

Theorem 1 If $b = s_1 \dots s_r$ is a strategy and b' is a strategy obtained from b by switching two adjacent slices, s_i and s_{i+1}, then

$$c(b) < c(b') \quad \text{if and only if} \quad \phi^+(s_i) > \phi^+(s_{i+1})$$
$$c(b) = c(b') \quad \text{if and only if} \quad \phi^+(s_i) = \phi^+(s_{i+1}).$$

Proof: Let γ be the subsequence of b that precedes $s_i s_{i+1}$. The expected costs of $\gamma s_i s_{i+1}$ and $\gamma s_{i+1} s_i$ are divided into contributions from three mutually exclusive situations: (1) neither s_i nor s_{i+1} can be excavated because either an ancestor of each failed to open

its doors or a treasure was found in one of γ's slices, (2) only one of the two slices can be excavated, and (3) both slices can be excavated.

The expected costs of $\gamma s_i s_{i+1}$ and $\gamma s_{i+1} s_i$ are identical in the first two cases because changing the position of slices that are not excavated cannot change the expected cost of a strategy. In the third case, the expected costs are given by

$$\begin{aligned} c(\gamma s_i s_{i+1}) &= c(\gamma) + c(s_i) + q^+(s_i)c(s_{i+1}) \\ c(\gamma s_{i+1} s_i) &= c(\gamma) + c(s_{i+1}) + q^+(s_{i+1})c(s_i). \end{aligned}$$

The first equation stems from the assumption that s_i is excavated with certainty after γ is excavated and from the fact that slice s_{i+1} is excavated after s_i with probability $q^+(s_i)$. The probability of excavating s_{i+1} after s_i is $q^+(s_i) = p^-(s_i) + p^0(s_i)$, and not just $p^0(s_i)$, because slice s_i is not on top of s_{i+1}. (Otherwise, b and b' could not both be strategies). The second equation holds by symmetry of i and $i+1$. The theorem follows by taking the difference between these two equations. □

The basis for our algorithm to find optimal excavation sequences lies in the observation that a slice with the highest ϕ^+ should be excavated immediately after the slice above it is excavated.

Theorem 2 If s_j is a slice with the highest ϕ^+ and s_j has an immediate parent s_i, then there exists an optimal strategy that includes the subsequence $s_i s_j$. If s_j is a top slice (root node), then there exists an optimal strategy that starts with s_j.

Proof: If s_i is an immediate parent of s_j, then s_i must be excavated before s_j. Suppose $s_i r_1 \ldots r_m s_j$ is a subsequence in some optimal strategy. Note that no r_k can be an ancestor of s_j because s_i is the immediate parent of s_j. Hence, we can repeatedly switch s_j with each r_i to obtain a new strategy in which s_j directly follows s_i. By Theorem 1, the cost of this strategy is less than or equal to the cost of the original. If s_j is a root node that follows $r_1 \ldots r_m$ in an optimal strategy, it can be switched to the front because no r_i can be its parent. Theorem 1 entails that the transformed strategy is at least as good as the original one. Hence, either s_j can start the strategy or immediately follow its parent. □

Theorem 2 implies that whenever a slice with the highest ϕ^+ is a top slice it can be placed first in a strategy. The remaining sequencing problem is smaller. If the best slice is not a top slice, it can be combined with its parent to form a single slice. Again, the remaining problem is smaller. Thus, each step reduces the number of slices by 1 until no slices are left and an optimal strategy is obtained. This algorithm is summarized in Figure 1.

Input: A collection of trees, with nodes N.

Output: An optimal search strategy stored in γ.

1. Set γ to the empty sequence.
2. Find a node $b \in N$ having the highest ϕ^+.
3. If b is a root node, then set $\gamma = \gamma b$ and remove b from N.
4. Otherwise, b has a parent b' in N. Combine nodes b' and b into a single node denoted by $b'b$. Place node $b'b$ in N and remove b' and b from N. Compute $\phi^+(b'b)$.
5. If some nodes are left in N, go to Step 2.

Figure 1: Algorithm to find optimal strategies.

It remains to explicate how to compute the cost and probabilities for the combined node, $b'b$. When pruning is not modeled, the parameters are computed by

$$\begin{aligned} c(b'b) &= c(b') + q^+(b') \cdot c(b) \\ p^+(b'b) &= p^+(b') + q^+(b') \cdot p^+(b), \end{aligned}$$

where b and b' are subsequences and not necessarily single nodes (Garey 1973). The expected cost of an optimal strategy is preserved by these transformations due to Eqs. (1) and (2).

However, when pruning is modeled, the combining equations depend on the topology of the tree. Suppose b' in the algorithm is the result of combining a subtree T' and b is the result of combining a subtree T. Since b' is a parent of b,

$$\begin{aligned} c(b'b) &= c(b') + \beta(b', b) \cdot c(b) \\ p^+(b'b) &= p^+(b') + \beta(b', b) \cdot p^+(b), \end{aligned}$$

where $\beta(b', b)$ is the probability that it is necessary to execute b after b' is executed. Moreover, $\beta(b', b)$ equals $\beta(r|s_1 \ldots s_z)$, where $b' = s_1 \ldots s_z$ and r is the first node in b.

The complexity of the algorithm is $O(n^2)$. On each iteration, finding the node with the highest ϕ^+ is $O(\log n)$ using a priority heap, and the calculation of p^+ and c for a merged node is $O(n)$. Since the algorithm iterates n times, the bound follows.

Our algorithm is similar to one described by Garey (1973). Both algorithms repeatedly transform a search tree by merging pairs of nodes into single nodes. They differ in the type of transformations applied; Garey's transformations always involve a leaf node while our transformation involves a node with the highest ϕ^+ value. Further, our algorithm deals with three possible outcomes of node exploration while Garey's deals with

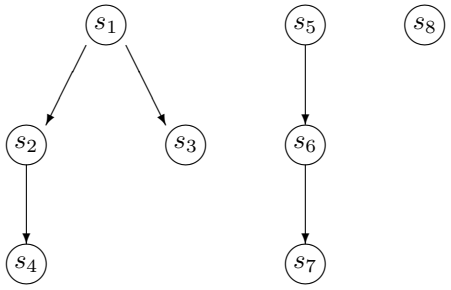

Figure 2: Search sites.

	c	p^+	p^0	ϕ^+
s_1	10	.1	.8	.01
s_2	5	.2	.5	.04
s_3	1	.8	0	.8
s_4	1	.7	0	.7
s_5	4	.1	.5	.025
s_6	9	.1	.7	.011
s_7	6	.2	0	.033
s_8	10	.2	0	.02

Table 1: Search sites parameters.

two. Nevertheless, Garey's algorithm can be amended to account for three-outcome evaluation as well and its complexity is the same as ours.

An Example

Consider three sites having the structure depicted in Figure 2 and the parameters given by Table 1. The optimal strategy for this example is calculated next using our algorithm.

Node s_3 has the highest ϕ^+. It is therefore combined with s_1. The parameters of the combined node are

$$\begin{aligned} c(s_1s_3) &= c(s_1) + p^0(s_1)c(s_3) \\ &= 10.8 \\ p^+(s_1s_3) &= p^+(s_1) + p^0(s_1)p^+(s_3) \\ &= .74 \end{aligned}$$

and, thus, $\phi^+(s_1s_3) = .069$. Now the node with the highest ϕ^+ is s_4. Its ϕ^+ is highest among all nodes including the newly created node s_1s_3. Hence, nodes s_4 and s_2 are combined and the resulting parameters are $c(s_2s_4) = 5.5$, $p^+(s_2s_4) = .55$, and $\phi^+(s_2s_4) = .1$. Node s_2s_4 now has the highest ϕ^+. It is therefore combined with its parent s_1s_3. The new parameters are

$$\begin{aligned} c(s_1s_3s_2s_4) &= c(s_1s_3) + p^0(s_1)q^+(s_3)c(s_2s_4) \\ &= 11.68 \\ p^+(s_1s_3s_2s_4) &= p^+(s_1s_3) + p^0(s_1)q^+(s_3)p^+(s_2s_4) \\ &= .828 \end{aligned}$$

and the resulting ϕ^+ is .071.

Node $s_1s_3s_2s_4$ is a root node and it has the highest ϕ^+. Thus, it is added to γ as a bloc. The next node is s_5. It is also a root node with the highest ϕ^+ and is therefore added to γ as a bloc. Now node s_7 is combined with its parent s_6. The resulting node s_6s_7 has a lower ϕ^+ value (0.018) than s_8. Thus, s_8 is the next bloc added to γ and s_6s_7 is the fourth and last.

Notably, any strategy produced by our algorithm consists of a sequence of blocs with decreasing ϕ^+ values. In this example, the blocs are $s_1s_3s_2s_4$, s_5, s_8, s_6s_7 with the respective decreasing ϕ^+ values, .071, .025, .02, and .018. This bloc structure coincides with Simon and Kadane's characterization of optimal strategies. It is not clear, however, whether this structure extends to strategies for OR graphs when three rather than two evaluation outcomes are possible.

The Dual Problem: AND Trees

We can think about search of OR trees as a procedure for proving the root node true: a node is proven true if and only if it is proven true by its own evaluation or at least one of its children is proven true. Proving true corresponds, in the gold-digging metaphor, to finding a treasure.

The task for AND trees is to prove the root node false. A node in an AND tree is proven false if and only if it is proven false by its own evaluation or at least one of its children is proven false.

The algorithm of Figure 1 with a minor change, switch the roles of p^- and p^+, finds optimal search strategies for AND trees.

Dynamic vs. Static Sequencing

Previous sections provide an algorithm that finds optimal search sequences for OR trees and for AND trees. In these two cases, optimal search sequences can be determined before search starts. However, this property does not hold in general. Next, we provide two examples where optimal sequences must be revised during search.

Consider the OR *graph* shown in Figure 3. There are three surface slices x, y, and z, and two deeper slices, v and w, each of which can be reached from two distinct surface slices. The rule is that a slice cannot be excavated until all of its parents are excavated. Thus, the graph encodes a partial order constraint on slice

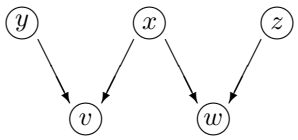

Figure 3: Sites with multiple paths.

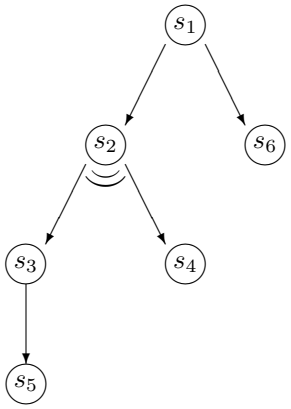

Figure 4: AND-OR tree.

excavations.[2]

Suppose x is a node with extremely high ϕ^+. Then x is excavated first. How should the rest of the nodes be ordered for excavation? If no treasure is found at x, then there are two options: (1) the digger learns with probability $p^-(x)$ that there is no gold beneath x in which case v and w will not be excavated, or (2) he learns with probability $p^0(x)$ that there may be gold in v and w. In the first case the decision about which slice to excavate next depends only on $\phi^+(y)$ and $\phi^+(z)$ while in the second case the decision depends on $\phi^+(yv)$ and $\phi^+(zw)$ as well. Hence, an optimal sequence cannot be determined until the result of excavating x is known, i.e., it must be determined dynamically.

Optimal searches of AND-OR trees require dynamic strategies as well. The interpretation of AND-OR trees is consistent with the definitions used previously for AND trees and OR trees. In particular, the tree of Figure 4 evaluates to true iff either node s_1, s_2, or s_6 evaluates true. Node s_2 (an AND node) evaluates to true iff s_2 is true or s_3 *and* s_4 evaluate to true. Node s_3 evaluates to true iff s_3 is true or s_5 is true.

Assume that the ϕ^+ values of $s_1 \ldots s_6$ are, respectively, 1, 900, 100, 200, 50, 130, that $\phi^+(s_3 s_4) = 150$, and that $\phi^+(s_4 s_5) = 110$. If one must commit to an execution order before search starts, then the choice would be either $s_1 s_2 s_3 s_4 s_6 s_5$ or $s_1 s_2 s_4 s_3 s_6 s_5$. Nodes s_2, s_3 and s_4 appear before s_6 because $\phi^+(s_2)$ and $\phi^+(s_3 s_4)$ are higher than $\phi^+(s_6)$ and s_6 is placed before s_5 because its ϕ^+ is higher.

The ordering between descendants of AND nodes is determined by their p^-/c ratios: those with higher values execute first. Assume for this example, that s_3 is executed before s_4 based on this criterion, i.e., the best a priori strategy is $s_1 s_2 s_3 s_4 s_6 s_5$.

Evaluating s_3 can produce three results: If s_3 evaluates true as expected, it is best to continue with the predetermined strategy. If s_3 evaluates false, then s_4 and s_5 are not evaluated and, hence, the expected cost of the remaining work is not effected by their relative location in the strategy. Otherwise, s_3 opens its doors and the strategy profits from a change: node s_6 should now be evaluated before node s_4, because its ϕ^+ value is higher than that of $s_4 s_5$. Thus, the best ordering of s_4 and s_6 is contingent on the results obtained by evaluating s_3.

Summary

We have presented an algorithm that finds optimal search strategies of AND trees and OR trees where pruning is modeled. Further, we have shown that optimal search strategies of AND-OR trees, AND graphs, and OR graphs cannot be represented as static permutations of the nodes. Consequently, to represent an optimal search strategy for these latter cases, one must construct a decision diagram that indicates the node to search next as a function of the outcomes of previous searches. The construction of such dynamic strategies is addressed by Slagle (1964) assuming there are no constraints on the order of node examination. Finding optimal search strategies subject to order constraints remains an open problem.

References

Garey M.R. 1973. Optimal task sequencing with precedence constraints. *Discrete Mathematics* 4:37–56.

Simon H.A., and Kadane, J.B. 1975. Optimal problem-solving search: all-or-none solutions. *Artificial Intelligence* 6:235–247.

Slagle, J.R. 1964. An efficient algorithm for finding certain minimum-cost procedures for making binary decisions. *Journal of the Association for Computer Machinery* 11:253–264.

[2] In the gold-digging metaphor this assumption is made to (say) prevent the collapse of a parent slice on its child if the child were excavated first.