A WRINKLE ON SATISFICING SEARCH PROBLEMS[1]

Jeffrey A. Barnett and Don Cohen
USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 90291

Abstract

The problem of optimally ordering the execution of independent disjuncts is explored. Only a single answer is sought, not necessarily the best one. By definition, this is called satisfying search. Since the disjuncts are independent, the total combined probability that a solution is found does not depend on the execution order. However, the ordering does affect the total expected execution time because execution ceases as soon as any solution is discovered. Therefore, the optimal ordering is the one that minimizes the total expected work. The new result is an algorithm to find this optimal ordering when the effects of executing a disjunct must be undone before another one can be tried. The algorithm is shown to have time complexity 0(n log n), where n is the number of disjuncts. This is the same complexity as for the original problem where undo times are ignored.

Introduction

Many investigators have examined problems of satisficing search: try the available methods one at a time until one of them satisfies the stated criteria, then stop. The objective is to find a method ordering with the least expected cost to solve the problem. Typically, only the probability of success and the expected cost are known for each method.

Method i is pairwise preferred to method j if, given only these two methods, it is less expensive to try i first. Pairwise preference is transitive. Therefore, if the optimal ordering of n methods is m1m2...mn and mn+1 is added, it is merely inserted somewhere in original ordering — all original methods stay in the same position relative to one another.

Below, the original problem is generalized: Associated with each method is a cost that must be paid, after trying the method, if another method is to be used. For example, the cost may be the time to undo the changes to the problem-solving state so that another method can be executed in the proper context.

The pairwise preference relation is no longer transitive and the simple insertion scheme is lost for the generalized problem. However, the criteria for optimal ordering is straightforward to derive. An algorithm that finds the optimal ordering is given, and it is shown to be of the same time complexity as the one for the original problem, namely o(n log n).

[1]This research is supported by the Defense Advanced Research Projects Agency under Contract No. MDA903 81 C 0335. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them.

The Original Problem

A set of methods is available, each of which has the potential to solve the same given problem. The methods can be applied to the problem in any order; however, they may only be tried one at a time. If one of the methods solves the problem, the remaining untried methods need not be used. In other words, only one solution is desired or necessary, and there is no interest in extra solutions nor any other results that might be produced by method execution.

The usual statement of problems in this class assumes that the probability that a particular method is successful and the execution cost of the method are independent of the order of execution and whether or not any other method is successful. Without this independence assumption, there is no general optimal ordering because the tradeoff between higher probability of success and lower expected cost is an application-dependent issue; the most general result possible then, is a partial ordering for method execution. However, with the independence assumption it follows that the total probability that at least one method will find a solution is independent of the order in which the methods are tried. Therefore, the residual problem is to determine the ordering with least expected cost.

A typical example of this class is the following: Let p be the probability that method i solves the stated problem and define q = l-p.. Further, let *c.* be the expected cost of trying method i. For example, c = p.s + q u . where s is the expected cost when successful and u is the expected cost when unsuccessful. What is the best order in which to apply a given set of methods to find a solution with the least expected cost?

The answer is simple:[2] Define p. = p /c.. Apply first the method with largest p; if it fails, try the method with the next largest p, etc. The order of application among methods with the same p value is immaterial to the total expected cost of finding a solution.

Two features of this result are noteworthy. First, a merit score (namely p. = p./c) can be calculated for a method independent of what other methods exist. Thus, if a new method becomes available, it can be evaluated separately and inserted into the current ordering of previously available methods with the assurance that the new ordering is optimal. Second, as a consequence, the pairwise preference ordering is transitive: Method i is *preferred* to method j if, given only methods i and j, the expected cost of trying i

[2]See Simon, H. A., and J. B Kadane, "Optimal Problem-Solving Search: All-or-None Solutions," *Artificial Intelligence* 6 (1975), 235-247 for this and other relat results.

before j is less than trying j first, i.e., $\varphi_i > \varphi_j$. This preference relation is surely transitive: If i is preferred to j and j is preferred to k, then i is preferred to k.

Next, a slight generalization of this problem is considered where both the simple insertion property and transitivity disappear.

A Generalization

As in the original problem, p_i is the probability that method i will solve the stated problem, $q_i = 1-p_i$, and c_i is its expected cost. In addition, let d_i be the cost incurred if method i fails and another method must be tried. For instance, if c_i is interpreted as the time necessary to determine whether the i'th medication combats a disease, then d_i can be interpreted as the *detoxification* time necessary before a different medication can be tested. Another interpretation of d_i is the expected cost to undo the effects of executing method i on the problem-solving state so that another method can be tried. The goal in this problem is to determine the correct medication, if any (or simply solve the problem), in the least expected time. For the original problem, all $d_i = 0$.

An Example

Define C_{ij} as the expected cost of (1) trying method i, (2) if it fails, waiting for time d_i, then (3) trying method j. It is evident that $C_{ij} = c_i + q_i(d_i + c_j)$ and that method i is preferred to method j if $C_{ij} < C_{ji}$. Suppose three methods are available with these values of p, c, and d — e and Φ are defined below.

i	p_i	c_i	d_i	e_i	Φ_i
1	.4	9	14.6	17.76	.0225
2	.5	10	18.9	19.45	.0257
3	.6	15	20.0	23.00	.0261

Then the six possible values of C_{ij} are

$$C_{12} = 23.76 \qquad C_{21} = 23.95$$
$$C_{23} = 26.95 \qquad C_{32} = 27.00$$
$$C_{31} = 26.60 \qquad C_{13} = 26.76$$

Thus, method 1 is preferred to method 2 ($C_{12} < C_{21}$), method 2 is preferred to method 3 ($C_{23} < C_{32}$), but method 3 is preferred to method 1 ($C_{31} < C_{13}$). Therefore, this example shows that the pairwise preference order is not transitive.

Formal Definitions

Though the pairwise preference relation does not induce a total ordering on the set of methods, the optimal ordering can be determined in the same order of time as for the original problem, o(n log n), where a simple sort on φ is sufficient. Some notation is necessary to develop this result. Let S be a sequence of methods and define P_S as the probability that at least one method in S solves the problem; $Q_S = 1-P_S$. Let C_S be the expected cost of trying the methods in S in the order mentioned (halting if one of them succeeds). Finally, D_S is the detoxification time necessary if all the methods in S fail and additional methods are to be tried.

Let Ø be the null sequence and i any method, then $P_\emptyset = 0$, $Q_\emptyset = 1$, $C_\emptyset = 0$, and $D_\emptyset = 0$; $P_i = p_i$, $Q_i = q_i$, $C_i = c_i$, and $D_i = d_i$. The following recurrence relations hold where S and T are sequences of methods and ST is the sequence formed by concatenating these two.

$$\begin{aligned}
P_{ST} &= 1 - Q_{ST} \\
Q_{ST} &= Q_S Q_T \\
C_{ST} &= C_S + Q_S[D_S + C_T] \quad T \neq \emptyset \\
&= C_S \quad T = \emptyset \\
D_{ST} &= D_T \quad T \neq \emptyset \\
&= D_S \quad T = \emptyset
\end{aligned}$$

It is easy to see that these definitions are associative, i.e., $X_{(ST)U} = X_{S(TU)}$ where X = P, Q, C, or D.

Optimal Ordering

The criterion for ordering nonfinal methods is straightforward to develop using these definitions. Let S, T, U, V be any sequences of methods such that V ≠ Ø.

$$\begin{aligned}
C_{STUV} &= C_{S(TU)V} \\
&= C_S + Q_S[D_S + C_{(TU)V}] \\
&= C_S + Q_S[D_S + C_{TU} + Q_{TU}[D_{TU} + C_V]]
\end{aligned}$$

If i and j are methods, then $C_{SijV} \leq C_{SjiV}$ is the criterion that i come before j. The expansion above, first with T = i and U = j, then with T = j and U = i, allows the transformation

$$C_{SijV} \leq C_{SjiV}$$
$$C_{ij} + Q_{ij}D_{ij} \leq C_{ji} + Q_{ji}D_{ji}$$

because all other terms cancel and $Q_{ij} = Q_{ji}$. Now, this form can be elaborated using the above definitions, then simplified as follows:

$$\begin{aligned}
c_i + q_i[d_i + c_j] + q_i q_j d_j &\leq c_j + q_j[d_j + c_i] + q_j q_i d_i \\
(1 - q_i)c_j + q_i(1 - q_j)d_i &\leq (1 - q_j)c_i + q_j(1 - q_i)d_j \\
p_j c_i + q_i p_j d_i &\leq p_i c_j + q_j p_i d_j \\
p_j / (c_j + q_j d_j) &\leq p_i / (c_i + q_i d_i).
\end{aligned}$$

It is convenient to define $e_i = c_i + q_i d_i$ and to note that e_i is the total expected time, including detoxification, to try method i when it is in any nonfinal position in the ordering. (For sequences S and T, the recurrence relation is $E_{ST} = E_S + Q_S E_T$ and in terms of C, Q, and D the relation is $E_S = C_S + Q_S D_S$.) Thus, the criterion that method i come before method j is

$$p_j / e_j \leq p_i / e_i$$

Since the steps in the derivation are reversible, this is both a necessary and sufficient condition. It is convenient to define $\Phi_i = p_i/e_i$ and note that Φ plays the same role here for nonfinal methods that φ did for all methods in the original problem.

The optimal ordering tries the nonfinal methods in order of decreasing Φ because Φ_i depends only on the i'th method and clearly induces a total ordering on these methods. Among methods with equal Φ, the ordering is immaterial. The Φ_i can be calculated and the methods sorted on its value in time 0(n log n).

However, the final method is not necessarily the one with lowest Φ. Let $m_1 \ldots m_n$ be an ordering consistent with $\Phi_i \geq \Phi_{i+1}$. Then for all except one method, say method j which should be final, the remaining methods are in this order. Therefore, the optimal ordering is the one of the form $m_1 \ldots m_{j-1} m_{j+1} \ldots m_n m_j$ for which C is minimized.

Let $S = m_1 \ldots m_{j-1}$ and $T = m_{j+1} \ldots m_n$ such that T ≠ Ø. Then the optimal ordering is the one that maximizes $C_{SjT} - C_{STj} > 0$. If all differences are nonpositive, the original sorted ordering is optimal.

This difference expands as

$$\begin{aligned} C_{S_iT} - C_{ST_j} &= C_S + Q_S[D_S + C_{j_T}] - [C_S + Q_S[D_S + C_{T_j}]] \\ &= Q_S[C_{j_T} - C_{T_j}] \\ &= Q_S[c_j + q_j[d_j + C_T] - [C_T + Q_T[D_T + c_j]]] \\ &= Q_S[P_T c_j + q_j d_j - [p_j C_T + Q_T D_T]]. \end{aligned}$$

For all j, the differences are calculated and the maximal one selected in time o(n) by the algorithm described next.

The Algorithm

The algorithm in Figure 1 is written in SIMULA as the class, *method_ordering.* There are *n* methods stored in the array *m.* Each method has the defined attributes *id* (a method identifier) and p, c, and *d* as described above. The derived attributes of a method are q, e. and *phi,* where *phi = p/e.* The procedure, *sort_on_phi,* is not shown explicitly; it may be any sorting algorithm that orders *m* on nonincreasing values of *phi* in time o(n log n).

Procedure *order* uses *sort_on_phi* then finds an optimal ordering. The steps are (1) find Q_S for each $S = m_1 \ldots m_i$ by noting that $Q_S = q_1 * \ldots * q_i$, (2) starting with $T = m_n$, iterate backwards until $T = m_2 \ldots m_n$ and find the maximal difference. and (3) if the maximal difference is positive, rearrange m into the optimal ordering. In all cases. $D_T = d_n$. However. Q_T and C_T must be updated using $Q_{jT} = q_j Q_T$ and $C_{jT} = c_j + q_j(d_j + C_T)$ so the iteration simulates $T := jT$ at each step.

If *order* is applied to the numerical example above, these steps occur.

1. The methods are sorted into the order {321} by their Φ values.
2. Two iterations are performed with the result that *dif* = .076 when *j - 2* and *dil* - -.25 when \ = 1. These iterations represent, respectively, the ordenngs {312} and {213}.
3. Since the maximal difference occurs when *j* = 2 and is positive, *m* is rearranged into the optimal ordering {312}.

For reference, the six values of C_{ijk} are

C_{321} = 32.580 C_{231} = 32.750
C_{312} = 32.504 C_{123} = 33.930
C_{213} = 32.830 C_{132} = 33.960

Conclusion

It is noted above that the transitivity property of the pairwise preference relation is lost in the generalized problem. Further, the simple insertion property is lost too. In the original problem, a new method could be evaluated separately, i.e., φ = p/c is calculated and does not depend on what other methods exist, then the new methdo* is inserted into the existing optimal ordering of other methods so that p values are nonincreasing. The new ordering is then known to be optimal. In the new problem, this is not possible because the current optimal ordering may have an arbitrary method as the final method.

The reason that both transitivity and the simple insertion •property are lost is easy to see — the expected cost of a method *depends* on its place in the ordering: If a method is in a nonfinal position, its cost is e_j but if it is the final method, its cost is c.

Another possible generalization of this class of ordering problems suggests itself; suppose the detoxification time between method i and method j is d.., i.e., d depends upon both the preceding and succeeding methods. Now the computation of the optimal ordering becomes at least as hard as the version of the traveling salesman problem where the salesman must visit each city once but does not need to return to his starting point. To see this, assume that p and c are the same for all n methods and q = 1-p is very nearly equal 1. Then the cost of an ordering $j = j_1 \ldots j_n$, where j is a permutation of the first n natural numbers is

$$\begin{aligned} C_j &= c[1 + q + q^2 + \ldots + q^{n-1}] \\ &\quad + qd_{j_1 j_2} + q^2 d_{j_2 j_3} + \ldots + q^{n-1} d_{j_{n-1} j_n} \\ &\approx nc + \Sigma d_{j_i j_{i+1}}. \end{aligned}$$

The approximation is justified since q is very nearly equal to 1. Minimizing C_j reduces to minimizing the summation over the permutations. j, and this is the traveling salesman problem.

```
CLASS method_ordering(n); INTEGER n;
BEGIN
      CLASS method(id,p,c,d); TEXT id;
            REAL p,c,d;
      BEGIN REAL PROCEDURE q; q:=1-p;
            REAL PROCEDURE e; e:=c+q*d;
            REAL PROCEDURE phi; phi:=p/e;
      END   method;

      REF(method) ARRAY m[1:n];

      PROCEDURE sort_on_phi;;

      PROCEDURE order;
      BEGIN REAL ARRAY qs[0:n];
            INTEGER j,best_j;
            REAL dif,best_dif,qt,ct,dt;
            REF(method) x;
            sort_on_phi;
            qs[0]:=1.0;
            FOR j:=1 STEP 1 UNTIL n
            DO qs[j]:=qs[j-1]*m[j].q;
            qt:=m[n].q;
            ct:=m[n].c;
            dt:=m[n].d;
            FOR j:=n-1 STEP -1 UNTIL 1 DO
            BEGIN dif:=qs[j-1]
                        *((1-qt)*m[j].c
                          +m[j].q*m[j].d
                          -(m[j].p*ct+qt*dt));
                  IF dif>best_dif THEN
                  BEGIN best_dif:=dif;
                        best_j:=j;
                  END;
                  ct:=m[j].c+m[j].q*
                      (m[j].d+ct);
                  qt:=m[j].q*qt;
            END;
            IF best_dif>0.0 THEN
            BEGIN x:-m[best_j];
                  FOR j:=best_j+1
                        STEP 1 UNTIL n
                        DO m[j-1]:-m[j];
                  m[n]:=x;
            END;
      END   order;
END   method_ordering;
```

Figure 1: SIMULA program that finds the optimal ordering