MANAGEMENT SCIENCE
Vol. 18, No. 7, March, 1972
*Printed in U.S.A.*

# A PROCEDURE FOR COMPUTING THE $K$ BEST SOLUTIONS TO DISCRETE OPTIMIZATION PROBLEMS AND ITS APPLICATION TO THE SHORTEST PATH PROBLEM*

EUGENE L. LAWLER

*University of California, Berkeley*

A general procedure is presented for computing the best, 2nd best, $\cdots$ , $K$th best solutions to a given discrete optimization problem. If the number of computational steps required to find an optimal solution to a problem with $n$ (0, 1) variables is $c(n)$, then the amount of computation required to obtain the $K$ best solutions is $O(Knc(n))$.

The procedure specializes to published procedures of Murty and of Yen for the assignment problem and the shortest path problem, respectively. A method is presented for reducing the required amount of storage by a factor of $n$, compared with the algorithms of Murty and of Yen. It is shown how the $K$ shortest (loopless) paths in an $n$-node network with positive and negative arcs can be computed with an amount of computation which is $O(Kn^3)$. This represents an improvement by a factor of $n$, compared with Yen's algorithm.

## 1. Ranking of Solutions

We seek to compute the best, second best, $\cdots$ , $K$th best solutions to a given discrete optimization problem. This may be desired, for example, because there are some particularly complex or subtle sets of conditions which are too difficult to incorporate directly into the constraints of the problem. By putting these conditions aside, we may be able to compute successively less desirable solutions until the best solution which does satisfy the conditions is found.

Assume, for convenience, that solutions to the optimization problem are expressed in terms of several (0, 1) variables, and that there exists an efficient computational procedure for determining optimal solutions, subject to the condition that certain of these variables are assigned fixed values, e.g., $x_1 = 0$, $x_3 = 1$.

One type of problem which meets these requirements is the assignment problem, where the (0, 1) variables are of the form $x_{ij}$, where $i, j = 1, 2, \cdots, n$. Murty [3] has described a procedure for computing the best, second best, $\cdots$ , $K$th best solutions to the assignment problem, which he refers to as "ranking" the solutions—a term we shall adopt. The basic computational procedure described here specializes to Murty's method.

Another problem which satisfies the necessary requirements is that of ranking "loopless" paths between two designated nodes of a network (i.e., paths with no repeated nodes). A procedure for this problem has been proposed by Yen [5], and the basic computational procedure described here also specializes to Yen's method.

## 2. Basic Computational Procedure

Assume that the optimization problem is one of minimization, and that the variables of the problem are $x_1, x_2, \cdots, x_n$. If a feasible solution does not exist for certain fixed values of the variables, the value of an optimal solution is taken to be $+\infty$.

The following simple procedure ranks solutions from the first to the $K$th, for predetermined $K$.

* Received November 1970; revised February 1971.

*Step* 0 (*Start*). Compute an optimal solution, without fixing the values of any variables, and place this solution in LIST as the only entry. Set $k = 1$.

*Step* 1 (*Output kth Solution*). Remove the least costly solution from LIST and output this solution, denoted $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)})$, as the $k$th solution.

*Step* 2 (*Test k*). If $k = K$, stop; the computation is completed.

*Step* 3 (*Augmentation of LIST*). Suppose, without loss of generality, that $x^{(k)}$ was obtained by fixing the values of $x_1, x_2, \cdots, x_s$. Leaving these variables fixed as they are, create $n - s$ new problems by fixing the remaining variables as follows:

$$\begin{aligned}
&(1)\ x_{s+1} = 1 - x_{s+1}^{(k)},\\
&(2)\ x_{s+1} = x_{s+1}^{(k)},\ x_{s+2} = 1 - x_{s+2}^{(k)},\\
&(3)\ x_{s+1} = x_{s+1}^{(k)},\ x_{s+2}^{(k)} = x_{s+2}^{(k)},\ x_{s+3} = 1 - x_{s+3}^{(k)},\\
&\ \vdots \qquad\quad \vdots\\
&(n - s)\ x_{s+1} = x_{s+1}^{(k)},\ x_{s+2} = x_{s+2}^{(k)},\ \cdots,\ x_{n-1} = x_{n-1}^{(k)},\ x_n = 1 - x_n^{(k)}.
\end{aligned}$$

Compute optimal solutions to each of these $n - s$ problems and place each of the $n - s$ solutions in LIST, together with a record of the variables which were fixed for each of them. Set $k = k + 1$. Return to Step 1.

The key to this procedure is the "branching" operation performed in Step 3. Let $X$ denote the set of feasible solutions for the problem for which $x^{(k)}$ is optimal, and let $X^{(1)}, X^{(2)}, \cdots, X^{(n-s)}$ denote the sets of feasible solutions for problems (1), (2), $\cdots$, $(n - s)$ created in Step 3. The reader can verify that $X^{(1)} \cup X^{(2)} \cup \cdots \cup X^{(n-s)} = X - \{x^{(k)}\}$. That is, the branching operation excludes $x^{(k)}$, and only $x^{(k)}$, from further consideration.

## 3. Computational Complexity of Basic Procedure

Suppose the number of steps required to compute a solution to a single optimization problem with $n$ variables is $c(n)$. Then the number of computational steps required to rank the $K$ best solutions is $O(Knc(n))$, i.e., of order $Knc(n)$. As we shall see, this bound may be improved in the case of the shortest path problem.

The number of entries recorded in LIST cannot exceed $K(n - 1)$. And, as Yen [5] has observed in regard to his method for shortest paths, one need maintain only the $K - k$ least costly solutions in LIST after the $k$th best solution has been outputted.

## 4. Modification to Reduce Storage

Even in the case where $K$ is not known in advance, the size of LIST can be restricted to $k$ entries after the $k$ best solution is outputted. This is accomplished by storing in LIST only the least costly of the $n - s$ solutions computed in Step 3 of the basic procedure. When this solution is eventually removed from LIST the same $n - s$ solutions are recomputed, and the next least costly among them is placed in LIST.

This procedure in effect requires Step 3 to be performed twice—once for the solution $x^{(k)}$ and once for the solution $x^{(l)}$, $l < k$, which gave rise to $x^{(k)}$ at a previous execution of Step 3. The amount of computation is effectively doubled, but since this is a change by only a linear scale factor, the overall computation remains $O(Knc(n))$.

The significant fact is that the modified procedure results in a net increase of one entry in the size of LIST at each iteration, so that LIST contains only $k$ entries after the $k$th solution is outputed.

The modified procedure is summarized as follows:

*Step* 0 (*Start*). Compute an optimal solution, without fixing the values of any variables. Output this solution, denoted $x^{(1)} = (x_1^{(1)}, x_2^{(1)}, \cdots, x_n^{(1)})$, as the least costly solution. LIST is empty. Set $k = 1$. Go to Step 3.2.

*Step* 1 (*Output kth solution*). Remove the entry $(x^{(j)}, y^{(j)})$ from LIST for which $y^{(j)}$ is least costly. Set $x^{(k)} = y^{(j)}$ and output $x^{(k)} = (x_1^{(k)}, x_2^{(k)}, \cdots, x_n^{(k)})$ as the $k$th solution.

*Step* 2 (*Test* $k$). If $k = K$, stop; the computation is completed.

*Step* 3 (*Augmentation of LIST*).

3.1. Suppose, without loss of generality, that $x^{(j)}$ was obtained by fixing the values of $x_1, x_2, \cdots, x_r$. Leaving these variables fixed as they are, create $n - r$ problems by fixing the remaining variables as follows:

$$\begin{aligned}
(1)\ x_{r+1} &= 1 - x_{r+1}^{(j)}, \\
(2)\ x_{r+1} &= x_{r+1}^{(j)},\ x_{r+2} = 1 - x_{r+2}^{(j)}, \\
&\vdots \\
(n-r)\ x_{r+1} &= x_{r+1}^{(j)},\ x_{r+2}\, x_{r+2}^{(j)},\ \cdots,\ x_{n-1} = x_{n-1}^{(j)},\ x_n = 1 - x_n^{(j)}.
\end{aligned}$$

Compute optimal solutions to each of these $n - r$ problems and from among the $n - r$ solutions find the least costly solution $y^{(j)}$ which exceeds $x^{(k)}$ in cost. Place $(x^{(j)}, y^{(j)})$ in LIST, together with a record of the variables which were fixed for $x^{(j)}$ and for $y^{(j)}$.

3.2. Suppose, without loss of generality, that $x^{(k)}$ was obtained by fixing the values of $x_1, x_2, \cdots, x_s$. Leaving these variables fixed as they are, create $n - s$ new problems by fixing the remaining variables as follows:

$$\begin{aligned}
(1)\ x_{s+1} &= 1 - x_{s+1}^{(k)}, \\
(2)\ x_{s+1} &= x_{s+1}^{(k)},\ x_{s+2} = 1 - x_{s+2}^{(k)}, \\
(3)\ x_{s+1} &= x_{s+1}^{(k)},\ x_{s+2} = x_{s+2}^{(k)},\ x_{s+3} = 1 - x_{s+3}^{(k)}, \\
&\vdots \\
(n-s)\ x_{s+1} &= x_{s+1}^{(k)},\ x_{s+2} = x_{s+2}^{(k)},\ \cdots,\ x_{n-1} = x_{n-1}^{(k)},\ x_n = 1 - x_n^{(k)}.
\end{aligned}$$

Compute optimal solutions to each of these $n - s$ problems and from among the $n - s$ solutions find the least costly solution $y^{(k)}$. Place $(x^{(k)}, y^{(k)})$ in LIST, together with a record of the variables which were fixed for $x^{(k)}$ and for $y^{(k)}$. Set $k = k + 1$. Return to Step 1.

## 5. Ranking of Shortest Paths

One of the more significant applications of the procedure is to the ranking of the $K$ shortest paths between two designated nodes of a network. We consider the case in which only loopless paths, i.e., paths without repeated nodes, are permitted. This is the case studied by Yen [5], as opposed to that dealt with by Dreyfus [1], in which repeated nodes are permitted.

The network is assumed to be in the form of a directed graph in which each arc $(i, j)$ is assigned a length $a_{ij}$. (Although we may permit negative arc lengths, we do assume there are no circuits with negative lengths.) Let the arcs of the network be numbered 1, 2, 3, $\cdots$, and for a given path let

$$\begin{aligned}
x_j &= 1 \quad \text{if arc } j \text{ is contained in the path,} \\
&= 0 \quad \text{otherwise.}
\end{aligned}$$

Suppose that it is desired to rank the $K$ shortest paths from node 0 to node $n$. We propose to arrange it so that the variables whose values are fixed to 1 are identified with arcs in a path from node 0 to another node. This is accomplished as follows.

Suppose, without loss of generality, that the arcs are numbered in such a way that the $k$th path contains arcs 1, 2, 3, $\cdots$, $m$ in sequence from node 0 to node $n$. Suppose, moreover, that the $k$th path is the shortest path subject to the conditions

$$(*) \qquad \begin{aligned} x_1 = x_2 = \cdots = x_p &= 1, \\ x_{r+1} = x_{r+2} = \cdots = x_q &= 0. \end{aligned}$$

In Step 3 of the basic computational procedure, $m - p$ new problems, each of the same form as $(^*)$, are created by fixing the remaining variables as follows:

$$(**) \qquad \begin{array}{ll} (p) & x_{p+1} = 0, \\ (p+1) & x_{p+1} = 1,\, x_{p+2} = 0, \\ & \vdots \\ (m-1) & x_{p+1} = x_{p+2} = \cdots = x_{m-1} = 1,\, x_m = 0. \end{array}$$

When the values of the variables are fixed in this way, the variables fixed to 1 will indeed be identified with arcs in a path from node 0 to another node. (Note that not all the cases dealt with in Step 3 of the general procedure need be considered, inasmuch as some of them would demand subsets of arcs not contained in any path.)

Now consider the problem of finding the shortest path from node 0 to node $n$ subject to conditions $(^*)$ above. Suppose arc $p$ is directed into node $p$. The desired path will consist of the given path from 0 to $p$, together with a path from $p$ to $n$. This latter path from $p$ to $n$ must not contain any of the arcs $r + 1$, $r + 2$, $\cdots$, $q$, and it must not pass through any of the nodes in the given path from 0 to $p$. Moreover, it must be the shortest possible path subject to all these conditions.

The desired shortest path from $p$ to $n$ can be computed as follows. Delete from the network all nodes in the given path from 0 to $p$, except node $p$. (In deleting a node, one deletes all arcs incident to it.) Also delete the arcs $r + 1$, $r + 2$, $\cdots$, $q$. Then simply compute the shortest path from $p$ to $n$ in the resulting network.

In a network with $n$ nodes, no path can contain more than $n - 1$ arcs, and not more than $n - 1$ problems can be created in Step 3 of the general procedure. If the network contains only positive arcs, an $O(n^2)$ computation is required for each of these problems, and if both positive and negative arcs are permitted, an $O(n^3)$ computation is necessary. It follows, as Yen observed, that the computation of the $K$ shortest paths is either $O(Kn^3)$ or $O(Kn^4)$, depending upon the type of network.

In the next section we show how the computation can be reduced to $O(Kn^3)$, in all cases.

## 6. Improvement in Efficiency

A computational procedure for computing the lengths of shortest paths between all pairs of nodes in a network has been attributed by Dreyfus [1] to Floyd [2] and Warshall [4]. The following procedure is that of Floyd, except that $u_{ij}^{(m)}$ is defined in terms of excluding nodes 1 to $m$, rather than including only those nodes. In particular, let

$u_{ij}^{(m)}$ = the length of a shortest path from $i$ to $j$ subject to the condition that it does not pass through nodes 1, 2, $\cdots$, $m$, then

$$\begin{aligned} u_{ij}^{(n)} &= \text{the length } a_{ij} \text{ of the arc } (i, j), \text{ if there is such an arc;} \\ &= +\infty \text{ otherwise} \end{aligned}$$

and

$$u_{ij}^{(m-1)} = \min \{u_{ij}^{(m)},\, u_{im}^{(m)} + u_{mj}^{(m)}\}.$$

The shortest paths for the complete set of problems (**), as defined in the previous section, can be computed as follows. Suppose, without loss of generality, that arc $i$, $i = 1, 2, \cdots$, is directed into node $i$. Let

$$\begin{aligned} v_i &= \text{the sum of the lengths of arcs } 1, 2, \cdots, i, \\ &= \text{the length of the fixed path, from node 0 to node } i. \end{aligned}$$

Delete nodes $0, 1, \cdots, p - 1$, and arcs $r + 1, r + 2, \cdots, q$. Compute $u_{jn}^{(i)}$ for all $i$, $j \geqq p$. The value of an optimal solution to problem (i) of (**) is

$$\min_{j>i+1} \{v_i + a_{ij} + u_{jn}^{(i)}\}.$$

The computation of the $u_{jn}^{(i)}$ values is $O(n^3)$. Given the values of $u_{jn}^{(i)}$, the computation for problem (i) of (**) is $O(n)$. Hence the overall computation in Step 3 is $O(n^3)$, and the computation of the $K$ shortest paths is $O(Kn^3)$.

(It is recognized that knowing the length of a shortest path is not the same as knowing the path itself, in terms of the specific arcs it contains. However, in the tradition of dynamic programming, we assume that the actual paths can be determined with no great difficulty.)

## References

1. Dreyfus, S. E., "An Appraisal of Some Shortest Path Algorithms," *Operations Research*, Vol. 17 (1969), pp. 395–412.
2. Floyd, R. W., "Algorithm 97, Shortest Path," *Comm. ACM*, Vol. 5 (1962), p. 345.
3. Murty, K. G., "An Algorithm for Ranking All the Assignments in Increasing Order of Cost," *Operations Research*, Vol. 16 (1968), pp. 682–687.
4. Warshall, S., "A Theorem on Boolean Matrices," *J. ACM*, Vol. 9 (1962), pp. 11–12.
5. Yen, J. Y., "Finding the *K* Shortest Loopless Paths in a Network," *Management Science*, Vol. 17, No. 11 (July 1971).