Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

# Letter to the Editor—The Multidimensional Assignment Problem

William P. Pierskalla,

# Letters to the Editor

## THE MULTIDIMENSIONAL ASSIGNMENT PROBLEM

**William P. Pierskalla**

*Case Western Reserve University, Cleveland, Ohio*

(Received June 1967)

The multidimensional assignment problem is a higher dimensional version of the standard (two-dimensional) assignment problem in the literature. The higher dimensions can be thought of as time or space dimensions or both. An algorithm is proposed for the solution of the multi-index assignment problem. The algorithm is based on a tree search technique of the branch-and-bound variety. It uses dual subproblems to provide easily computed bounds for the primal assignment problem.

IN RECENT years much interest has concentrated on integer linear programming problems and in particular several papers have focused on the zero-one integer *LP* problem. This paper is also concerned with the zero-one integer problem but particularly in regard to the special case of an optimal assignment in more than two dimensions.

We begin by describing the three-dimension assignment problem and later give its multidimensional statements. The three-dimensional assignment problem is the problem of assigning one item to one job at one point or interval in time in such a way as to minimize the total costs of the assignment. Mathematically, if it is assumed that $p \leqq q \leqq r$ then we want to minimize:

$$\sum_{i=1}^{i=p} \sum_{j=1}^{j=q} \sum_{k=1}^{k=r} c_{ijk}\, x_{ijk}, \tag{1}$$

subject to

$$\left.\begin{aligned} \sum_{i=1}^{i=p} \sum_{j=1}^{j=q} x_{ijk} &\leqq 1 \quad \text{for} \quad k=1, \cdots, r. \\ \sum_{i=1}^{i=p} \sum_{k=1}^{k=r} x_{ijk} &\leqq 1 \quad \text{for} \quad j=1, \cdots, q, \\ \sum_{j=1}^{j=q} \sum_{k=1}^{k=r} x_{ijk} &= 1 \quad \text{for} \quad i=1, \cdots, p, \end{aligned}\right\} \tag{2}$$

$$x_{ijk}=0, 1 \quad \text{for all} \quad i, j, k. \tag{3}$$

$c_{ijk}$ is the cost of assigning item $i$ to location $j$ at time $k$. $x_{ijk}=1$ means the $i$th item is assigned to the $j$th location at time $k$. $x_{ijk}=0$ means the $i$th item is not assigned to location $j$ at time $k$. Equality holds in the third set of constraints since it is presumed that a total assignment of the $p$ items is required.

As an example of a three-dimensional assignment problem consider the case where a company realizes that its distribution system is in need of two new warehouses in the next three years and a third new warehouse in about five years. The company is not only faced with the problem of where to locate the warehouses but also when should the warehouses be built. Eight different cities are being considered as possible sites for the three warehouses. The company wishes to select the combination of warehouse, site, and time so as to minimize the total costs associated with this expansion. For example, $c_{ijk}$ would be the total cost of locating warehouse $i$ on site $j$ at time $k$; $c_{ijk}$ could include discounted construction costs, future transportation costs for shipping into and out of this location, operating costs, etc.

Another example could be the launching of $x$ weather satellites into $y$ directional orbits at $z$ different atmospheric levels from the earth's surface. This example could include a fourth dimension if we considered the various possible times of launch.

The general idea behind the multidimensional assignment problem is that there are often additional scheduling dimensions besides just scheduling men to jobs that should be taken into consideration in making the optimal allocation. These additional dimensions may be time, or space, or some other factors perhaps qualitative in nature.

A more general formulation of this problem as a three-dimensional transportation type problem has been considered by E. SCHELL[11] and K. B. HALEY[6] with the exception that they do not consider constraint (3) but rather $x_{ijk} \geqq 0$. Since the polyhedron defined by (2) and $x_{ijk} \geqq 0$ has in general a great number of noninteger extreme points, the approach taken by Schell and Haley will most often yield a noninteger solution (for large $p$, $q$, and $r$). A similar comment applies to the simplex method when (3) is replaced by $x_{ijk} \geqq 0$.

Before proceeding further we will list some standard definitions.

1. A solution $x = (x_{ijk})$ is called feasible if $x$ satisfies (2) and (3).
2. A solution $x$ is called optimal if $x$ is feasible and $x$ minimizes (1).

In the case where $p = q = r$, we have the three-dimensional assignment problem where all of the constraints (2) are equality constraints.

## METHOD OF SOLUTION

THE METHOD of solution is merely a form of implicit enumeration of all basic feasible points. For example, if we let $p = 3$ and $q = r = 4$ we can sketch the tree of feasible solutions (Fig. 1).

Each node of the tree is characterized by a triple of integers $i$, $j$, $k$. Thus if we are at node 132 then $i = 1$, $j = 3$, $k = 2$ and $x_{132} = 1$, $x_{1jk} = 0$ for $j \neq 3$ or/and $k \neq 2$. Denote by *level* ($t$) the set of all nodes of the tree such that the $i$ index has value $t$. In Fig. 1 level (1) would be the set of nodes {111, 112, 113, 114, 121, 122, 123, 124, 131, 132, 133, 134, 141, 142, 143, 144}.

Actually in level (1) of the figure there are nine branches from each of the sixteen nodes. Only one complete set of the nine branches has been drawn. Similarly level (2) has four branches from each node and we have only sketched one full node. Thus for this particular example there are 576 feasible paths. For the general case $p \leqq q \leqq r$ the total number of paths is given by the product of the permutations $P_p{}^q \cdot P_p{}^r$.

The algorithm proceeds by selecting a feasible solution $\bar{x}$ and computing its objective function value $\bar{z} = c \cdot \bar{x}$. We then turn to a dual problem to compute easily the lower bounds for $p$ primal subproblems. (These $p$ primal subproblems are defined later in this section.)

Now each primal subproblem has a dual that we will call its dual subproblem The solutions to the $p$ dual subproblems provide $p$ lower bounds $l_1, \cdots, l_p$ for the possible values of the primal objective function as we proceed down the tree of feasible solutions. For example, at level (1) of the tree if $\bar{z} = l_1$, the first lower bound, then we know that $\bar{x}$, is an optimal solution, and we may terminate provided that only one optimal solution is desired.

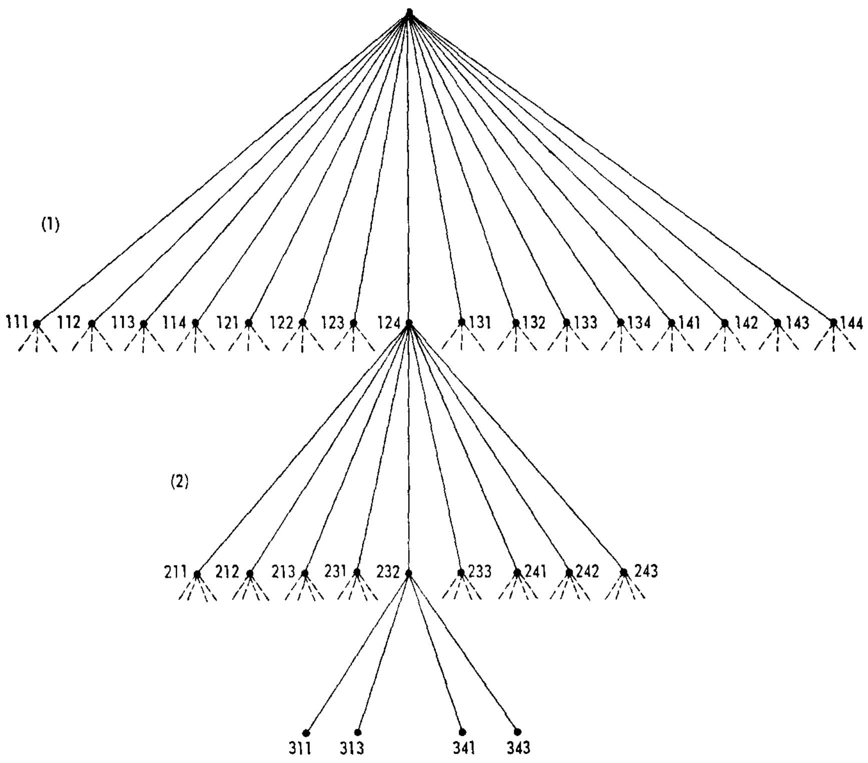

**Fig. 1.** Partial tree graph of the feasible solutions to $p=3$, $q=4$, $r=4$.

If $\bar{z} > l_1$ then we proceed systematically down the tree using $l_i$ at level ($i$) in order to evaluate whether it might be profitable to continue the search further.

This method of solution is similar to the branch-and-bound method used by LITTLE, MURTY, SWEENEY, AND KAREL[9] in their approach to the traveling salesman problem. At each node of the tree of feasible solutions they have calculated a lower bound for proceeding further on any of the outgoing branches from that node. If the lower bound exceeds the lowest value of the known feasible solutions to the problem then a better solution cannot be obtained by proceeding further from that node. On the other hand if the lower bound is strictly less than the lowest value of the known feasible solutions, then a better feasible solution might be found by proceeding further from that node. In their algorithm just as in the algorithm proposed in this paper, the lower bounds are easily calculated but do not represent optimal solutions to the bounding problems. Thus the lower bounds are not as tight as possible but computer time is saved in their speed of calculation. In this regard the algorithm of BALAS[1] for the general 0–1 linear integer programming problem also uses easily calculated but not necessarily tight lower bounds. The Balas algorithm, however, differs slightly from the algorithm in this paper because of the fact that all $2^n$ possible solutions are (implicitly) enumerated. This is necessary since he is solving the general 0–1 problem and does not know in advance the structure of the set of feasible solutions.

Before going further with a more explicit description of the algorithm we will state some interesting and pertinent results in lemma and theorem form. Proofs of these results are available in the appendix.

LEMMA 1.

(*1*) *Feasible integer solutions to* (2) *and* (3) *exist.*

(*2*) *Furthermore a vector $x$ is a feasible integer solution to* (2) *and* (3) *if and only if* (*a*) *there are precisely $p$ coordinates of $x$ with value one and the remaining coordinates have value zero and* (*b*) *the $p$ elements with value one consist of $x_{ijk}$'s with the property that the $i$ subscripts are a permutation of the integers* $1, \cdots, p$, *the $j$ subscripts are a permutation of $p$ of the integers* $1, \cdots, q$, *and the $k$ subscripts are a permutation of $p$ of the integers* $1, \cdots, r$.

For example, the solution:

$$x_{iii}=1 \quad \text{for all} \quad i=1, \cdots, p,$$
$$x_{ijk}=0 \quad \text{otherwise,}$$

is a feasible integer solution.

THEOREM 1.

(*1*) *An optimal solution to* (1), (2), *and* (3) *exists although not necessarily uniquely.*

(*2*) *When $t=1$ an upper bound on $\sum\sum\sum c_{ijk}\, x_{ijk}$ is given by*

$$M_t=\sum_{i=t}^{i=p} \max_{k,j} \{c_{ijk}\}. \qquad (4)$$

(*3*) *When $t=1$ a lower bound on $\sum\sum\sum c_{ijk}\, x_{ijk}$ is given by*

$$m_t=\sum_{i=t}^{i=p} \min_{k,j} \{c_{ijk}\}. \qquad (5)$$

Define the *ith primal subproblem* as the multi-index assignment problem remaining after the first $i-1$ assignments have been made but without regard as to which $j$ and $k$ indices have been assigned. For example, in the previous graph, Fig. 1, if assignment $x_{121}=1$ and all other $x_{1jk}=0$ then the 2nd subproblem would be to consider the assignment problem for $p=2$, $q=4$, and $r=4$ given by

$$\min \sum_{i=2}^{i=3} \sum_{j=1}^{j=4} \sum_{k=1}^{k=4} c_{ijk}\, x_{ijk},$$

subject to

$$\sum_{i=2}^{i=3} \sum_{j=1}^{j=4} x_{ijk} \leqq 1 \quad \text{for} \quad k=1, \cdots, 4,$$
$$\sum_{i=2}^{i=3} \sum_{k=1}^{k=4} x_{ijk} \leqq 1 \quad \text{for} \quad j=1, \cdots, 4,$$
$$\sum_{j=1}^{j=4} \sum_{k=1}^{k=4} x_{ijk}=1 \quad \text{for} \quad i=2, 3,$$

and $x_{ijk}=0, 1\ \forall i=2, 3, j$ and $k=1, \cdots, 4$ where $c_{ijk}$ are given in the original (0th subproblem) problem.

LEMMA 2.

*The numbers $m_t$ given by* (5) *are lower bounds for the optimal policy for the t-th subproblem,* $t=1, \cdots, p$.

Many lower bounds of the $m_t$ variety can be formed by considering feasible solutions to the dual problem generated by each primal subproblem where we do not consider the primal integer constraints. However, if an optimal solution to the dual problems generated by each primal subproblem is found and if we denote these optimal solutions by $d_t$ for $t=1, \cdots, p$ then clearly (from the proof of lemma 2) $d_t \geqq m_t$ and the $d_t$ provide the best lower bound that can be achieved by considering the dual subproblems.

We are now able to state the algorithm for finding the optimal solution to (1), (2), and (3). In the algorithm we denote the lower bounds for the $t$th subproblem by $l_t$. In the computer runs given in the last section of this paper the $l_t$ are the $m_t$ of (5); however, if $d_t$ is known, we would set $l_t=d_t$.

Define the sets $J_i$ and $K_i$ as the sets of all $j$ and $k$ indices respectively that correspond to the values of $j$ and $k$ at each node visited in levels (1) through and including level ($i$). For example, in Fig. 1 if we were at node 232 in level (2) then $J_2=\{2, 3\}$ and $K_2=\{4, 2\}$. Also define $\tau^i_{jk} \equiv$ lower bound on the total cost of proceeding all the way down the tree through node $ijk$. $\tau^i_{jk}$ is composed of the actual cost down the tree graph following a specific path to node $ijk$ plus the lower bound cost of proceeding from node $ijk$ to the bottom of the tree graph. Specifically

$$\tau^i_{jk} \equiv \tau^{i-1}_{j_{i-1}k_{i+1}} - l_i + c_{ijk} + l_{i+1},$$

where we define $\tau^0_{j_0k_0} \equiv l_1$. The notation $i-1, j_{i-1}, k_{i-1}$ denotes the immediately preceding node visited before arriving at node $ijk$. Thus $\tau^{i-1}_{j_{i-1}} - l_i$ is the actual total cost of a specific path down to node $i-1, j_{i-1}, k_{i-1}$ but not beyond; $c_{ijk}$ is the actual cost of going to node $ijk$ from some immediately preceding node such as $i-1, j_{i-1}, k_{i-1}$; and $l_{i+1}$ is the lower bound on the cost of proceeding from node $ijk$ on to the end of the tree graph.

## ALGORITHM

**1.** Compute the first lower bound $l_1$; if the solution $x$ that corresponds to the lower bound $l_1$ is feasible, then set $\bar{x} \leftarrow x$ and $\bar{z} \leftarrow c\bar{x}$ and terminate; an optimal solution has been found. If $x$ is not feasible go to **2.**

**2.** Choose a feasible solution $\bar{x}$, say $\bar{x}_{iii}=1$, $\bar{x}_{ijk}=0$ otherwise. Set $\bar{z} \leftarrow c \cdot \bar{x}$. If $\bar{z}=l_1$, then terminate; an optimal solution has been found. If not, compute $l_i$ for $i=2, 3, \cdots, p$, set $i \leftarrow 0$ and go to step **3.** The algorithm now iterates on $k$, then on $j$, and finally on $i$ until the entire tree graph has been (implicitly) searched.

**3.** Set $i \leftarrow i+1$. If $i=p+1$ then we had just previously arrived at the bottom of the tree graph and it is necessary to back up to the node visited just prior to visiting the last level [level ($p$)]. This backing up is accomplished by setting $i \leftarrow i-2$, $j \leftarrow j_i$ and $k \leftarrow k_i$, in that order. Go to step **7.** If $i<p+1$ then set $j \leftarrow q+1$ and $k \leftarrow r+1$, and go to step **4.**

**4.** Set $j \leftarrow j-1$. Go to step **5.**

**5.** If $j\epsilon J_{i-1}$, then go to step **4**. In this case the $j$ element under consideration has already been included in the path and cannot be brought in again (*see* Lemma 1). If $j\notin J_{i-1}$ and $j=0$. then go to step **6**; else go to step **7**.

**6.** Since $j=0$ then we have exhausted all of the $j$ index numbers 1, 2, $\cdots$, $p$ for the given value of $i$ hence we must back up to the last previously visited node in level $(i-1)$ by setting $i\leftarrow i-1$, $j\leftarrow j_i$, and $k\leftarrow k_i$. If now $i=0$ then terminate; the current candidate for optimality $\bar{x}$ is the optimal solution since we have exhausted all paths in the tree graph of feasible solutions. If $i>0$ then go to step **7**.

**7.** Set $k\leftarrow k-1$; go to step **8**.

**8.** If $k\epsilon K_{i-1}$ then go to step **7**. If $k\notin K_{i-1}$ and $k=0$ then set $k\leftarrow r+1$ and go to step **4**; else go to step **9**.

**9.** Since $i\neq 0, j\neq 0, k\neq 0, j\notin J_{i-1}$, and $k\notin K_{i-1}$ we have an acceptable node to test. Form

$$\tau_{jk}^{i}\leftarrow\tau_{j_{i-1}\,k_{i-1}}^{i-1}-l_i+c_{ijk}+l_{i+1}.$$

If $\tau_{jk}^{i}\geq\bar{z}$, then a better feasible solution cannot be found by going further from node $ijk$; hence go to step **7**. If $\tau_{jk}^{i}<\bar{z}$, then perhaps a better feasible solution can be found. Set $j_i\leftarrow j$, $k_i\leftarrow k$, and $\tau_{j_ik_i}^{i}\leftarrow\tau_{jk}^{i}$. Denote by $x(j_i, k_i)$ the path down to node $i, j_i, k_i$ and the path from node $i, j_i, k_i$ suggested by the bound $l_{i+1}$. If $x(j_i, k_i)$ is a feasible path then set $\bar{x}\leftarrow x(j_i, k_i)$ and $\bar{z}=c\cdot\bar{x}$ since we have found a feasible path with a lower objective function value. Now if $\bar{z}=l_1$ terminate; an optimal solution has been found. If either $\bar{z}>l_1$ or $x(j_i, k_i)$ is not feasible go to step **10**.

**10.** Set $J_i\leftarrow J_{i-1}\cup\{j_i\}$ and $K_i\leftarrow K_{i-1}\cup\{k_i\}$. Go to step **3**.

The algorithm terminates whenever:

(1) $\bar{z}=l_1$ in which case $\bar{x}$ is optimal where $\bar{z}=c\cdot\bar{x}$ or

(2) in the case that $i=0$ there are no more iterations to consider, then $\bar{x}$ is optimal.

It is obvious that the algorithm is finite since no branch of the tree is searched more than once and there are a finite number of branches.

It should be noted that the process can be stopped at any time and the current stored $\bar{z}$ and its corresponding $\bar{x}$ is the best feasible solution found up to that time.

This method systematically searches the tree of feasible paths; however, it abandons a branch of the tree whenever it becomes obvious that a better solution cannot be achieved by pursuing that branch further. Thus if the sum of all feasible $c_{ijk}$'s down to a certain branch point plus the minimum bound $l_{i+1}$ to proceed further exceeds the value $\bar{z}$ of the current candidate $\bar{x}$ for optimality then an optimal path cannot be found by going further down that branch. This last statement is given in the form of a theorem.

THEOREM 2. *In the above algorithm if* $\tau_{j_tk_t}^{t}>\bar{z}$ *then an optimal solution to* (1), (2), *and* (3) *cannot be found on the branch of the solution tree where we fix* $(0, j_0, k_0)$, $(1, j_1, k_1)$ $\cdots$ $(t, j_t, k_t)$.

Furthermore, it should be noted that all optimal solutions may be found by not rejecting paths when $\tau_{jk}^{i}=\bar{z}$ but only when $\tau_{jk}^{i}>\bar{z}$. Also adding or subtracting a constant amount $M$ from each $c_{ijk}$ only changes the value of the optimal solution by $p\cdot M$ and does not change the optimal solution vector. Therefore, if $c_{ijk}>0$ for all $i, j, k$ the problem could be changed to

$$\min\sum\sum\sum(c_{ijk}-M)x_{ijk},$$

subject to (2) and (3) where we let $M=\min_{i,j,k} c_{ijk}$. Such a change would reduce the values of the $c_{ijk}$'s and for hand computations at least makes the problem easier to handle.

Moreover, if some assignment $x_{ijk}=1$ is not possible from physical considerations, the algorithm need not be modified but only set $c_{ijk}=Q$ where $Q$ is some very large positive number. And in the context of the warehouse location example (or any other example), it is not necessary that the time periods $k=1, \cdots, r$ be evenly spaced periods of equal length. If two warehouses but no more could be built in, say, the first year, then let $k=1$ be a period of length 1 week, $k=2$ a period of length 51 weeks, and define $x_{ijk}$ as the decision to begin construction of warehouse $i$ at location $j$ at the *beginning* of period $k$. In this way it would be possible to build two warehouses in the first year but not more than two warehouses, since period 3 starts after the first year has ended.

Finally, we can say that at each branch of the tree it would be possible to find a lower bound or set of lower bounds peculiar to that specific branch. Thus the bounds would most likely be tighter and termination would be achieved sooner. These bounds could be obtained by considering the dual problem defined by a $t_{jk}$th primal subproblem where not only the previously assigned $i$ indices were removed but also the previously assigned $j$ and $k$ indices. We could then generate lower bounds of the form of lemma 2 for these new subproblems. Unfortunately this method has the great disadvantage that the bounds would have to be computed at each node of the tree reached by the algorithm, whereas the bounds in lemma 2 only need to be computed once at the beginning of the problem.

The multi-index formulation of the problem is: if $p_1 \leqq p_2 \leqq \cdots \leqq p_n$ then we wish to minimize

$$\sum_{i_1, \cdots, i_n} c_{i_1, \cdots, i_n} x_{i_1, \cdots, i_n},$$

subject to

$$\begin{aligned}
\sum_{i_1, \cdots, i_{n-1}} x_{i_1, \cdots, i_n} &\leqq 1, & \forall i_n &= 1, \cdots, p_n,\\
&\cdots & &\cdots\\
\sum_{i_1, i_3, \cdots, i_n} x_{i_1, \cdots, i_n} &\leqq 1, & \forall i_2 &= 1, \cdots, p_2,\\
\sum_{i_2, \cdots, i_n} x_{i_1, \cdots, i_n} &= 1, & \forall i_1 &= 1, \cdots, p_1,\\
x_{i_1, \cdots, i_n} &= 0, 1, & \forall i_1, &\cdots, i_n.
\end{aligned}$$

The foregoing lemmas and theorems as well as the algorithm for solving the multi-index problem generalize in the obvious way. However, the dimensionality of the problem increases extremely fast, namely by products of factorials, and it is very doubtful that even small multidimensional problems (dimension $\geqq 5$) could be solved in any reasonable amount of time.

## COMPUTATIONAL EXPERIENCE WITH THE ALGORITHM

USING A Univac 1107 we applied the algorithm to several small three-dimensional problems. The values of $c_{ijk}$ were obtained from a pseudorandom number generator where the random numbers were all rounded to integers between 00 and 99.

| Problem no. | $p$ | $q$ | $r$ | Total no. of variables | Approximate no. of feasible paths | Machine running time,[a] sec |
|---|---|---|---|---|---|---|
| 1 | 4 | 4 | 4 | 64 | 576 | 1 |
| 2 | 5 | 5 | 5 | 125 | 14,400 | 3 |
| 3 | 6 | 6 | 6 | 216 | $5.2 \times 10^5$ | 9 |
| 4 | 8 | 8 | 8 | 512 | $1.6 \times 10^9$ | 173 |
| 5 | 8 | 8 | 8 | 512 | $1.6 \times 10^9$ | 151 |
| 6 | 4 | 5 | 6 | 120 | $4.3 \times 10^4$ | 2 |
| 7 | 6 | 7 | 8 | 336 | $1.0 \times 10^8$ | 9 |
| 8 | 4 | 8 | 8 | 256 | $2.8 \times 10^6$ | 3 |
| 9 | 3 | 7 | 9 | 189 | $1.0 \times 10^5$ | 1[b] |
| 10 | 8 | 10 | 12 | 960 | $3.6 \times 10^{13}$ | 128 |

[a] Does not include the compilation time of approximately 10 sec.
[b] In this problem the lower bound provided a feasible (hence optimal) solution.

Unfortunately, as in branch-and-bound techniques in general, the algorithm is slow if the bounds are very loose, if the current candidates for the optimal policy at each step decrease the objective function slowly, and the optimal policy occurs near the end of the search. In this case the suboptimal method presented in reference 10 yields a faster program.

## APPENDIX

### PROOFS

*Lemma 1* part (1) is obvious. Part (2) ($\rightarrow$). Let $x$ be a feasible solution to (2) and (3).

Since $\sum_j \sum_k x_{ijk}=1 \forall i=1, \cdots, p$ then upon adding these $p$ equality constraints

$$\sum_{i=1}^{i=p} \sum_{j=1}^{j=q} \sum_{k=1}^{k=r} x_{ijk}=p.$$

But $x_{ijk}=0, 1 \forall i, j, k \rightarrow$ there are exactly $p$ of the $x_{ijk}$'s equal to one. Now assume that the $x_{ijk}$'s that equal one do not satisfy part (*b*). Thus there is at least one $x_{ijk}=1$ with at least one common index element with another $x_{ijk}=1$, say $x_{i_0j_0k_0}=x_{i_1j_1k_0}=1$. But by (2) we must have $\sum_{i=1}^{i=p} \sum_{j=1}^{j=q} x_{ijk_0} \leqq 1$ since $x$ is feasible; thus we have a contradiction since $\sum_{i=1}^{i=p} \sum_{j=1}^{j=q} x_{ijk_0} \geqq 2$. ($\leftarrow$) Now (a) and (b) are given. Clearly (3) is satisfied and since each inequality of (2) can contain at most one $x_{ijk}=1$ and each equality must contain one $x_{ijk}=1$ because of the permuting of the indices then (2) holds. Hence by definition $x$ is feasible.

THEOREM 1. *We first prove part (3):*

*Part (3).*
$$\sum_{i=1}^{i=p} \sum_{j=1}^{j=q} \sum_{k=1}^{k=r} c_{ijk}x_{ijk} \geqq \sum_{i=1}^{i=p} \min_{j,k} c_{ijk} \sum_{j=1}^{j=q} \sum_{k=1}^{k=r} x_{ijk} = \sum_{i=1}^{i=p} \min_{j,k} c_{ijk} = m_1.$$

*Part (1). This part is obvious from part (3) above and lemma* 1.

*Part (2). This proof is* mutatis mutandis *the same as part (3) above.*

LEMMA 2. *The* $t$th *subproblem is given by:*

$$\sum_{i=t}^{i=p} \sum_{j=1}^{j=q} \sum_{k=1}^{k=r} c_{ijk}\, x_{ijk}, \qquad (6)$$

*subject to*

$$\left.\begin{array}{ll} \sum_{i=t}^{i=p} \sum_{j=1}^{j=q} x_{ijk} \leqq 1 & \forall k=1, \cdots, r, \\ \sum_{i=t}^{i=p} \sum_{k=1}^{k=r} x_{ijk} \leqq 1 & \forall j=1, \cdots, q, \\ \sum_{j=1}^{j=q} \sum_{k=1}^{k=r} x_{ijk} = 1 & \forall i=t, \cdots, p, \end{array}\right\} \tag{7}$$

$$x_{ijk}=0, 1. \tag{8}$$

The dual of this subproblem where we omit (8) and use instead $x_{ijk} \geqq 0$ is given by

$$\max \sum_{i=t}^{i=p} y_i - \sum_{i=p+1}^{p+q+r} y_i, \tag{9}$$

subject to:

$$-y_{p+j}+y_i-y_{p+q+k} \leqq c_{ijk}, \tag{10}$$

for all

$$i=t, \cdots, p,$$
$$j=1, \cdots, q,$$
$$k=1, \cdots, r,$$

$$\left.\begin{array}{l} y_i \text{ unconstrained for } i=t, \cdots, p, \\ y_i \geqq 0 \text{ for } i=p+1, \cdots, p+q+r. \end{array}\right\} \tag{11}$$

In (10) if we set $y_i=0$ for all $i=p+1, \cdots, p+q+r$ and set $y_i=\min_{j,k} \{c_{ijk}\}$ for $i=t, \cdots, p$ then both (10) and (11) are satisfied and from (9)

$$\sum_{i=t}^{i=p} y_i = m_t.$$

Hence this solution is feasible for this dual problem. Now by the standard results of linear programming if $x^*$ is optimal for (6), (7), and (8) and if $\bar{x}$ is optimal for (6), (7), and $x_{ijk} \geqq 0$ then $cx^* \geqq c\bar{x} \geqq \sum_{i=t}^{i=p} = m_t$.

THEOREM 2. *By lemma 2, the bounds $l_t$ are at least as small as the minimum cost solution to the* $t$th *subproblem. Thus if the total cost for the first $t-1$ branches plus the lower bound from $t$ on is greater than the current candidate for optimality, viz., $\bar{z}$, then a better path (i.e., lower cost) cannot be found from the* $t$th *level on.*

## REFERENCES

1. BALAS, EGON, "An Additive Algorithm for Solving Linear Programming with Zero-One Variables," *Opns. Res.* **13,** 517–545 (1965).
2. BALINSKI, M. L., "Integer Programming: Methods, Uses, Computation," *Management Sci.* **12,** 253–313 (1965).
3. GEOFFRION, A. M., "Integer Programming by Implicit Enumeration and Balas' Method," Rand Corporation Memorandum RM-4783-PR, 1966.
4. GLOVER, FRED, "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," *Opns. Res.* **13,** 879–919 (1965).
5. HALEY, K. B., "The Existence of a Solution to the Multi-index Problem," *Opnal. Res. Quart.* **16,** 471–474 (1965).
6. ———, "The Multi-index Problem," *Opns. Res.* **11,** 368–379 (1963).
7. ———, "The Solid Transportation Problem," *Opns. Res.* **10,** 448–463 (1962).
8. LAWLER, E. AND D. E. WOOD, "Branch-and-Bound Methods: A Survey," *Opns. Res.* **14,** 699–719 (1966).
9. LITTLE, J. D. C., K. G. MURTHY, D. W. SWEENEY, AND C. KAREL, "An Algorithm for the Traveling Salesman Problem," *Opns. Res.* **11,** 972–989 (1963).
10. PIERSKALLA, W. P., "The Tri-Substitution Method for Obtaining Near-Optimal

Solutions to the Three-Dimensional Assignment Problem," Tech. Memo. No. 71, Operations Research Group, Case Institute of Technology, Cleveland, Ohio, October, 1966.

11. SCHELL, E., "Distribution of a Product by Several Properties," *Proc. Second Symp. in Linear Programming*, Washington, D. C., January 27–29, Vol. 1–2, pp. 615–642, 1955.

# STOPPING RULES FOR QUEUING SIMULATIONS

**Irwin W. Kabak**

*New York University, New York, New York*

(Received March 8, 1967)

In queuing simulations when service times and/or inter-arrival times are exponentially distributed it is possible to obtain independent estimates of the quantities of interest, such as the probability of a request being served immediately or the proportion of requests that are delayed more than (say) $t$ time units. It is shown that a weighted average of estimates of the probability of being served immediately is asymptotically unbiased for two simple queuing systems; it is also shown that an unweighted average is biased for one of these systems. Because the estimates are independent, the calculation of the variance of the weighted average is simplified. Expressions are presented for the calculation of the mean and variance of the estimate of interest. This paper presents only the nucleus of an idea and indicates several areas where research should prove useful.

AN EVER-PRESENT question in all simulations is 'Have enough trials been processed by the simulator?' A usual stopping rule is: stop the simulation when the (theoretical) variance of the statistic that estimates the quantity of interest is within given limits. However, there is an operational difficulty in the administration of this measure, i.e., because of the correlation of successive trials one cannot easily calculate the required variance. (This is discussed in some detail by HAUSER, ET AL.[3] If one could avoid the entire problem of correlated estimates and obtain independent estimates, the calculation of the required variance could be done more readily.

## INDEPENDENT ESTIMATES FOR QUEUING SIMULATIONS

IT IS indeed fortunate for simulations of queuing systems that, under certain circumstances, independent estimates of the quantity of interest are available. COX AND SMITH[1] have introduced the concept of tours that we now present.

The *state* of the system is defined as the number of calls in the system either being served or waiting to be served. When an event (as arrival or a departure) causes the system to be in state $j$, we begin a *tour*. A departure from state $j$ and a subsequent return, of the same type (arrival or departure) that began the tour completes it. Referring to Fig. 1, all tours illustrated for state $C$ are [1, 3), [3, 7), [7, 10), [2, 4), and [4, 8). Note that [1, 2) is not a tour. In some queuing simulations cer-