NEW ALGORITHMS FOR THE BULK, ZERO-ONE TIME MINI-MAX TRANSPORTATION MODEL*

L.R. Foulds, University Of Canterbury
P.B. Gibbons, Massey University
New Zealand

SUMMARY

A survey of transportation problems is presented beginning with the classical Hitchcock-Koopmans cost minimization model. Each model is discussed and its best known solution procedures are referenced. One in particular, the bulk zero-one time-mini-max model is studied in depth. Two new algorithms for this model are outlined - one based on branch and bound enumeration and one on a backtracking technique. In addition an implementation of Bhatia's method, the only other existing algorithm for this model of which the authors are aware, is described. Computational experience gained from the use of all three algorithms is presented.

1. INTRODUCTION

The transportation problem is one of the classical deterministic O.R. models. It was first formulated by Hitchcock, but Koopmans was the first to notice how graph theory could be used to develop efficient solution techniques. For the original reference see Hitchcock [9]. Hence, it is usually called the Hitchcock-Koopmans model. This model, and the variations we develop later, find application in the situation in which it is desired to transport amounts of a single commodity from a number of sources to a number of destinations in a way which satisfies all destination demands but does not exceed source capacity. Objectives vary between models - typically one wishes to minimize total transportation costs or time. Problems of this nature occur frequently in the New Zealand industrial scene. As an example, de Pont and Barr [5] used a transportation problem algorithm to schedule the return of re-usable bottles from collection centres to bottling factories. They stated that annual savings of $50,000 for a cost of $1,000 consultant time were indicated.

This paper surveys the various transportation problems in existence, beginning with the classical Hitchcock-Koopmans model. Each model is analysed and efficient solution pro-
cedures for it are referenced. In addition two new algorithms for the bulk zero-one time-mini-max model are described and compared with an implementation of Bhatia's method, [2], currently the only other published procedure for solving this model of which the authors are aware.

2. REVIEW OF TRANSPORTATION MODELS

We shall use the following notation to define a model:

\[ m \] = the number of sources,
\[ n \] = the number of destinations,
\[ a_i \] = the capacity of source \( i \),
\[ b_j \] = the demand of destination \( j \),
\[ c_{ij} \] = the unit transportation cost from source \( i \) to destination \( j \),
\[ t_{ij} \] = the time taken to supply destination \( j \) from source \( i \), (which is assumed independent of the number of units supplied).

A solution to a model will be denoted by

\[ x_{ij} \] = the number of units supplied by source \( i \) to destination \( j \),
\[ y_{ij} \] = 1 if source \( i \) supplies destination \( j \), and 0 otherwise.

2.1 THE COST MINIMIZATION MODEL (HITCHCOCK-KOOPMANS)

In this model unit transportation costs are given between all source-destination pairs. There is no restriction on the number of sources which can supply a given destination. The objective is to minimize the total transportation cost. It is assumed that the total capacity of all sources equals the total demand of all destinations, i.e., \( \sum a_i = \sum b_j \). This condition can always be achieved by the introduction of 'dummy' sources or 'dummy' destinations. The model is:

\[
\begin{align*}
\text{Minimize} & \quad \sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} \\
\text{Subject to} & \quad \sum_{j=1}^{n} x_{ij} = a_i, \quad i = 1,2,\ldots,m, \\
& \quad \sum_{i=1}^{m} x_{ij} = b_j, \quad j = 1,2,\ldots,n, \\
& \quad x_{ij} \geq 0, \quad i = 1,2,\ldots,m, \quad j = 1,2,\ldots,n.
\end{align*}
\]

Glover, Karney, Klingman and Napier [6] have presented an in-depth computational comparison of the basic solution algorithms for solving transportation problems. The comparison is performed using computer codes for the dual simplex transportation method, the out-of-kilter method, the
primal simplex transportation method, and a large scale L.P. code. They state that the most efficient solution procedure arises by coupling a primal transportation algorithm (embodying recently developed methods for accelerating the determination of basis trees and dual evaluations) with a version of the row minimum start rule and a modified row first negative evaluator rule. The resulting method has been found to be 100 times faster than the L.P. code and 9 times faster than a streamlined version of the out-of-kilter code. The method's median solution time for solving problems with 1,000 sources and 1,000 destinations on a CDC 6600 is 17 seconds, with a range of 14 to 22 seconds.

2.2 THE BULK, ZERO-ONE TIME MINIMIZATION MODEL

In this model the cost of transportation from a source to a destination is independent of the number supplied - there is simply a fixed charge for each source - destination pair. Thus the model has bulk costs rather than unit costs. The authors prefer to refer to these costs as times, although there may exist realistic situations in which some other optimality criterion is appropriate. There is a further restriction in that each destination must obtain all of its demand from exactly one source. Hence the term "zero-one". The objective is to minimize the total time of all transportation. The model is:

\[
\begin{align*}
\text{Minimize} & \quad \sum_{i=1}^{m} \sum_{j=1}^{n} t_{ij} y_{ij} \\
\text{subject to} & \quad \sum_{j=1}^{n} b_{ij} y_{ij} \leq a_i, \quad i=1,2,\ldots,m, \\
& \quad \sum_{i=1}^{m} y_{ij} = 1, \quad j=1,2,\ldots,n, \\
& \quad y_{ij} = 0 \text{ or } 1, \quad i=1,2,\ldots,m, \quad j=1,2,\ldots,n.
\end{align*}
\]

De Maio and Roveda [4] discussed a real-world problem which gives rise to their formulating the above model. A company has to allocate all the items necessary for production and maintenance in its warehouses so as to minimize total delivery costs, where it is possible to omit fixed costs associated with each warehouse. The system comprises one source (from which the goods arrive), m warehouses and n shops. Each shop requires a certain number of items; the amount requested for every item is known. Every shop can be supplied by any warehouse and for the sake of organizational efficiency, every item must be allocated to one warehouse only. Although the problem is a multi-commodity one, it can be formulated as model (2.1). De Maio and Roveda have developed a search method to solve the model which somewhat resembles Balas's filter method. However, like
many implicit enumeration schemes their method requires a large amount of storage when implemented with a computer code. De Maio and Roveda wrote such a code for a UNIVAC 1108 in 1969 in FORTRAN V and managed to solve problems with up to $m = 19$, $n = 43$ in 15 seconds.

Srinivasan and Thompson \cite{13} presented a branch and bound procedure which solves the model by using the fact that an optimal solution for it can be characterized as a basic feasible solution to a slightly modified transportation problem. Such a solution can be obtained by an algorithm similar to the subtour elimination method for solving the travelling salesman problem. The procedure appears to be relatively more efficient when the number of destinations greatly exceeds the number of sources. Srinivasan and Thompson state they believe that their algorithm is computationally more efficient than the implicit enumeration approach, as it utilizes the underlying structure of the transportation problem. However, they do not present any computational experience to support this.

Murthy \cite{11} presented an algorithm based on lexicographic search. Solutions are represented as a string of numbers and are systematically generated in some hierarchy of their values. Bounds are generated for sets of strings and if a bound is found to be more than the value of a known feasible solution, its set can be discarded. The efficiency of the approach seems to be sensitive to the construction of a good trial solution and the availability of tight lower bounds. Murthy states that the storage requirements for his algorithm are far less than those required for the two previous algorithms. However, no computational experience is given.

2.3 THE BULK, TIME MINI-MAX

In this model, as in the previous one, there are bulk (rather than unit) transportation costs referred to as times. As with the first model there is no restriction on the number of sources which may supply a given destination. However, the objective is to minimize the maximum transportation time. This situation arises when it is assumed that all transportation begins simultaneously and the objective is to complete the delivery programme in the shortest possible time. Then that time is going to be the maximum time of those source-destination pairs actually used. The model is:

\[
\begin{align*}
\text{Minimize} & \quad \sum_{i=1}^{m} \mathbf{t}_{ij} x_{ij} > 0 \\
\text{subject to} & \quad \sum_{j=1}^{n} x_{ij} = a_i, \quad i=1,2,\ldots,m,
\end{align*}
\]
Applications of this model occur in the mangement of emergency facilities, such as fire engines and ambulances. The problem has been posed and discussed by Bhatia [2]. He presented an algorithm for it which uses as a subroutine a method to solve model (2.2) (e.g., any of the methods in [4], [11] or [13], could be used.)

3. A COMPARISON OF ALGORITHMS FOR THE BULK, ZERO-ONE, MINI­MAX TIME TRANSPORTATION PROBLEM

In this section we compare three algorithms for the solution of the Model (2.4). The first is an adaption of an algorithm developed by Murthy [11] for the solution of model (2.2) the second an implementation of an algorithm proposed by Bhatia [2], and the third a branch-and-bound procedure developed by the authors. We shall first describe each of these algorithms in detail, and then present a timing comparison for them as implemented in Burroughs Algol on a B6700. In the description of each of these algorithms, we shall adhere to the notation introduced in Section 2, including the matrices $T = (t_{ij})$ and $Y = (Y_{ij})$. To avoid multiple subscript levels, we shall at times refer to elements of a matrix or vector by use of bracketted subscript notation, rather than the usual subscript notation. For example, we shall sometimes refer to $t_{ij}$ as $T[i,j]$. In addition, we shall assume that the rows and columns of $T$ have been initially reordered, so that $a_i > a_{i+1}$ for $1 \leq i < m$ and $b_j < b_{j+1}$ for $1 \leq j < n$.

3.1 THE ADAPTED MURTHY ALGORITHM

Murthy's algorithm was designed to solve Model (2.2) and uses the well-known technique of backtracking [3][7]. Suppose we have a feasible solution $Y = y_{ij}$ to Model II. Then, since each column of $Y$ contains exactly one '1', we can equally well represent this solution by the vector $a = (a_1, a_2, \ldots, a_n)$, where $y_{aj} = 1$ for $j = 1, \ldots, n$. Such a vector can be regarded as a word of length $n$ with letters from the set $\{1, 2, \ldots, m\}$. The goal of Murthy's algorithm, then, is to construct, by backtracking, a word $a = (a_1, a_2, \ldots, a_n)$ such that

(i) $a$ represents a feasible solution to Model II, and
(ii) $z_0 = \sum_{j=1}^{n} T[a_j, j]$ is a minimum over all feasible solutions.

To improve the efficiency of the search, Murthy uses the following bounding procedure: Construct a matrix
AB = (ab_{ij}) such that

(i) \forall k \in \{1, \ldots, m\} i \in \{1, \ldots, m\} such that ab_{ij} = k (i, j = 1, \ldots, n)

and (ii) i < k \Rightarrow T[ab_{ij}, j] < T[ab_{kj}, j] (j = 1, 2, \ldots, n)

Now suppose we have constructed the partial word PS = (PS[1], \ldots, PS[r]) (r < n) representing a partial (feasible) solution in which the jth destination (1 \leq j \leq r) gets all its requirement from source PS[j]. Let a'_i (i = 1, \ldots, m) be the residual capacity of the source i after supplying as specified in PS. Now the kth destination (r < k \leq n) can obtain its supply from the source s only if b_k \leq a'_s. Hence for the block of solutions with PS as leader we have the following lower bound on their cost:

\[ LB(PS) = \sum_{j=1}^{r} R[PS[j], j] + \sum_{j=r+1}^{n} T[\beta_j, j] \]

where \( \beta_j = ab_{xj} \) and \( x = \min \{ i | b_{ij} \leq a'_i(ab_{ij}) \} \)

If LB(PS) > VT where VT is the cost of the incumbent feasible solution, then we can backtrack immediately, rather than attempting to extend PS to a complete solution. For Model (2.4) the cost of a solution \( \alpha = (a_1, a_2, \ldots, a_n) \) is \( Z_\alpha = \max_{1 \leq j \leq n} T[a_j, j] \). Hence, a lower bound of solutions based on \( PS = (PS[1], \ldots, PS[r] (r < n) \) would be

\[ LB(PS) = \max \{ \max_{1 \leq j \leq r} \{ T[PS[j], j] \}, \max_{r \leq j \leq n} \{ T[\beta_j, j] \} \} \]

where \( \beta_j \) is defined as above. An initial feasible solution may be found by using the procedure outlined by Bhatia [2].

3.2 Bhatia's Algorithm

Bhatia presented a procedure which begins by either identifying an initial feasible solution or indicating that no such solution exists. The initial solution is iteratively improved until an optimal solution is produced. The improvement process uses as a subroutine a simplified model (2.2) algorithm. Bhatia did not specify such an algorithm. This is surprising since the particular algorithm chosen will have a critical effect on the overall performance of his procedure. However, one approach could be to use a simplified version of Murthy's algorithm, described in Section 3.1. Our version of Bhatia's procedure adopts this approach.

3.3 A Branch-and-Bound Algorithm

A branch-and-bound procedure is frequently used in the search for optimal solutions to combinatorial problems. The reader is referred to Balas [1] and Lawler and Wood [10] for a detailed description of this procedure. In our case
we shall be constructing a binary decision tree with a lower bound associated with each node. Recalling that our claim is to construct a matrix \( Y = y_{ij} \) such that \( y_{ij} = 1 \) if source \( i \) supplies destination \( j \) and 0 otherwise, we shall 'fathom' each node \( \gamma \) by choosing a particular \( y_{ij} \) and then constructing a node \( \gamma_1 \) representing all solutions in which \( y_{ij} = 1 \), and a node \( \gamma_2 \) representing all solutions in which \( y_{ij} = 0 \). During the search process, entries in \( Y \) which have not been fixed will be set to tentative values which are subject to change. When an entry \( y_{ij} \) is fixed at 1 we shall subtract demand \( b_j \) from capacity \( a_i \). Also, during the search we shall keep track (as with the previous two algorithms) of an incumbent representing the lowest cost solution found so far. An initial incumbent (if any) may be obtained using the method described in Section 3.1. We commence by constructing a node representing all feasible solutions. A lower bound for this node is determined as follows:

\[
\forall j \in \{1, \ldots, n\} \text{ find } t_{kj} = \min_{1 \leq i < m} \{ t_{ij} \} \text{ and set } y_{kj} := 1.
\]

Set all remaining entries in \( Y \) to 0.

Our lower bound for this initial node is then \( \max_{1 \leq i < m, 1 \leq j < n} \{ t_{ij} | y_{ij} = 1 \} \).

We now fathom this node as follows. If its bound is not strictly less than the cost of the incumbent it is considered no further (i.e., it is "pruned" from the tree). Otherwise, if the matrix \( Y \) represents a feasible solution it is stored as the new incumbent, after which the node is pruned. Finally, if the matrix \( Y \) does not represent a feasible solution (and the bound is less than the incumbent cost) we must branch further.

Consider all sources for which the demand exceeds the capacity, i.e. the set \( I \) of "infeasible" rows \( i \) such that \( \sum_{j=1}^{n} y_{ij} b_j > a_i \) (note that we must exclude columns with a fixed 1 entry from this sum since their demands have already been deducted). For each row \( i \in I \) there exists at least one entry \( y_{ij} \) which must be changed from '1' to '0'. If this is done, another entry in the \( j \)th column must be set to '1' so that the \( j \)th destination can be supplied. In carrying out this change we are likely to incur the "penalty" of a possible increase in the lower bound. More specifically, if

\[
a_j = \min_{1 \leq k < m} \{ t_{kj} \mid k \neq i, \ a_k - b_j \geq 0 \}
\]

To be precise, we shall also have to exclude from consideration "fixed 0" entries in the \( j \)th row. If this means that there are no entries to be considered, we set \( a_j = \infty \), which effectively states that we cannot alter the value of \( y_{ij} \). Then our new lower bound will be max \{old bound, \( a_j \)\}. We
shall therefore refer to $a_j$ as the penalty for changing the entry in column $j$.

We now determine the column with the maximum penalty, i.e., if $S = \{k \mid i \in I : y_{ik} = 1\}$ and excluding "fixed 1" columns from this set, we find $j$ such that $a_j = \max_{k \in S} a_k$ (take the smallest $j$ in the event of a tie). Since this column will incur the greatest penalty if changed, we first investigate the possibilities that arise if the column is fixed. (An alternative strategy could be to first consider changing the column with the smallest penalty.) That is, we construct the node $y_1$ representing all solutions in which the entry $y_{ij}$ in column $j$ is fixed at value '1'. Seeing that entry $y_{ij}$ is fixed, another '1' entry in row $i$ must be changed. Thus our lower bound for node $y_1$ is $\max \{\text{bound for father node } k \in S \setminus \{j\} \mid a_k \mid y_{ik} = 1\}$. If there is no $k \in S \setminus \{j\}$ such that $y_{ik} = 1$ we set the bound to $\infty$ to indicate that $y_{ik}$ cannot be fixed at 1. We now deduct demand $b_j$ from capacity $a_i$, and, if $a_i$ is still $> 0$, proceed to fathom node $y_2$, in the same way as described above.

After fathoming node $y_1$, we prune $y_1$ from the tree, reset $y_{ij}$ to 0, add demand $b_j$ back to capacity $a_i$, and then construct node $y_2$ representing all solutions in which $y_{ij}$ is fixed at 0. Here we incur the penalty described above, so that the bound for node $y_2$ is $\max \{\text{bound for father node, } a_j\}$. We now set the entry $y_{kj}$ for which $t_{kj} = a_j$ (tentatively) to 1 and proceed to fathom node $y_2$. Having fathomed $y_2$ we reset $y_{kj}$ to 0, $y_{ij}$ to 1 and then prune $y_2$ from the tree. At this stage we have completed the fathoming of the father node $y$.

At the conclusion of this recursive procedure we print out the incumbent as the optimal solution.

The procedure is best illustrated by considering the example summarized in Table 1. In this example we have indicated an initial feasible solution by circling entries in the matrix $T$ which correspond to '1's in the matrix $Y$. It can be seen that the cost of this solution is 10. We

<table>
<thead>
<tr>
<th></th>
<th>$a_1$</th>
<th>$a_2$</th>
<th>$a_3$</th>
<th>$a_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>40</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>7</td>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$b_1$</th>
<th>$b_2$</th>
<th>$b_3$</th>
<th>$b_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>30</td>
<td>20</td>
<td>20</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 1: Example
now proceed with the branch-and-bound algorithm, and obtain the solution trees shown in Figure 1. At the node marked with an * the following feasible solution was obtained: $\gamma_{31} = \gamma_{12} = \gamma_{23} = \gamma_{44} = \gamma_{15} = 1$. This solution has cost 3, which is not greater than any bound in the tree, and thus represents an optimal solution.

Figure 1: Solution tree for Example

![Solution tree](image)

3.4 A COMPARISON OF THE THREE ALGORITHMS

As mentioned earlier, all three algorithms were implemented in Burroughs Algol on a B6700. In order to compare the efficiency of the algorithms, each algorithm was run on a number of randomly generated problems of selected sizes of $n$ and $m$. Before presenting the results of these tests, we shall describe the method used for this random generation. So that we can control the size of the problem, we first fix values of $m$ and $n$. We also decide on a maximum source capacity (maxcap) and a maximum time cost (maxtime). Using a random number generator we now randomly generate integers $a_i$, $i = 1,\ldots,m$ such that $1 \leq a_i \leq \text{maxcap}$. Now let totcap be the total capacity of all sources. Then, in order to ensure that the total demand does not exceed the total capacity, we generate the integers $b_j$, $j = 1,\ldots,n$ in the following way:

\[
\text{for } j := 1 \text{ step 1 until } n \text{ do begin }
\begin{align*}
\quad b_j &:= \text{a random integer } \in \{1, \ldots, \text{min}\{\text{maxcap, totcap}\}\}; \\
\quad \text{totcap} &:= \text{totcap} - b_j
\end{align*}
\text{end}
\]

The bound totcap in fact resulted in the generation of a large percentage of problems with no feasible solution, and so was later modified to $\text{min}\{\text{maxcap, totcap}\} * 0.8$. Finally, we sort the elements of the vectors $a$ and $b$ into decreasing order, and set $t_{ij} := \text{a random integer } \in \{1,\ldots, \text{maxtime}\}$ for $1 \leq i \leq m$, $1 \leq j \leq n$.

The results of these tests are presented in Table 2. From Table 2 it can be seen that there is very little
Table 2: Test Results

<table>
<thead>
<tr>
<th>m</th>
<th>n</th>
<th># problems generated</th>
<th>Branch-and-Bound Time (in seconds) Bhatia</th>
<th>Murthy</th>
<th>Branch-and-Bound Time (in seconds) Bhatia</th>
<th>Murthy</th>
<th>Range of Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>16</td>
<td>.06</td>
<td>.08</td>
<td>.05-.10</td>
<td>.05-.20</td>
<td>.05-.12</td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>13</td>
<td>.16</td>
<td>.35</td>
<td>.10-.50</td>
<td>.10-.68</td>
<td>.18-.35</td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>12</td>
<td>.20</td>
<td>.61</td>
<td>.12-.67</td>
<td>.18-.17</td>
<td>.32-.43</td>
</tr>
<tr>
<td>30</td>
<td>15</td>
<td>5</td>
<td>.27</td>
<td>1.47</td>
<td>.23-.28</td>
<td>1.45-1.48</td>
<td>1.68-1.85</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>2</td>
<td>.72</td>
<td>1.16</td>
<td>.47-.97</td>
<td>.93-1.38</td>
<td>1.05-2.25</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>4</td>
<td>.43</td>
<td>3.11</td>
<td>.32-.65</td>
<td>3.00-3.23</td>
<td>3.63-4.70</td>
</tr>
<tr>
<td>40</td>
<td>30</td>
<td>5</td>
<td>16.52</td>
<td>5.10</td>
<td>8.58</td>
<td>4.50-6.48</td>
<td>6.15-13.17</td>
</tr>
<tr>
<td>50</td>
<td>30</td>
<td>5</td>
<td>.72</td>
<td>6.71</td>
<td>8.39</td>
<td>6.57-6.97</td>
<td>7.77-10.00</td>
</tr>
<tr>
<td>60</td>
<td>30</td>
<td>5</td>
<td>.92</td>
<td>10.01</td>
<td>14.73</td>
<td>9.35-11.42</td>
<td>10.97-24.88</td>
</tr>
</tbody>
</table>

The difference between the algorithms of Bhatia and Murthy, although Bhatia's algorithm appears to become marginally better as m and n are increased. In all cases, however, except one (viz. m = 40, n = 30), the branch-and-bound algorithm performed significantly better than the other two. The poor performance for the case m = 40, n = 30 was caused by two generated problems which forced the algorithm to take 39.63 and 40.03 seconds respectively, whereas the remaining three generated problems were solved in 1.48, .93, and .55 seconds respectively. The existence of such "difficult" problems could be a potential drawback for the branch-and-bound method. It should also be noted that the branch-and-bound algorithm is implemented using a recursive procedure, and hence is more expensive in its use of storage than the relatively economical methods of Bhatia and Murthy.

REFERENCES


