Mathematical Programming 8 (1975) 91–103.
North-Holland Publishing Company

# A BRANCH AND BOUND ALGORITHM FOR THE GENERALIZED ASSIGNMENT PROBLEM*

G. Terry ROSS
*University of Massachusetts, Amherst, Mass., U.S.A.*

and

Richard M. SOLAND
*University of Texas, Austin, Tex., U.S.A.*

Received 18 September 1973
Revised manuscript received 15 October 1974

This paper describes what is termed the "generalized assignment problem". It is a generalization of the ordinary assignment problem of linear programming in which multiple assignments of tasks to agents are limited by some resource available to the agents. A branch and bound algorithm is developed that solves the generalized assignment problem by solving a series of binary knapsack problems to determine the bounds. Computational results are cited for problems with up to 4 000 0–1 variables, and comparisons are made with other algorithms.

## 1. Introduction

The purpose of the classical assignment problem and many variations on it is to find optimal pairings of agents and tasks. Each task is assigned to a single agent, each agent is given a single task, and the suitability of a particular set of assignments is determined by a single criterion function. The assignment of several tasks to a single agent is possible only by enlarging the problem to include fictitious tasks and duplicate agents. There is, however, no way to restrict these multiple assignments.

From an applications standpoint, a more useful model would allow the assignment of several tasks to a single agent, provided these tasks do not require more of some resource than is available to the agent. Prob-

* This research was partly supported by ONR Contracts N00014-67-A-0126-0008 and N00014-67-A-0126-0009 with the Center for Cybernetic Studies, The University of Texas.

lems that can be accurately represented by this "generalized" assignment model include assigning software development tasks to programmers, assigning jobs to computers in computer networks [1], scheduling variable length television commercials into time slots, and scheduling payments on accounts where contractual agreements specify "lump sum" payments. Other applications include fixed charge plant location models in which customer requirements must be satisfied by a single plant [7, 10], and communication network design models with certain node capacity constraints [9].

A mathematical formulation of the generalized assignment problem is:

$$\text{(P)} \qquad \text{minimize} \quad \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij}\,, \tag{1}$$

$$\text{subject to} \quad \sum_{j\in J} r_{ij}x_{ij} \leqslant b_i \quad \text{for all } i \in I\,, \tag{2}$$

$$\sum_{i\in I} x_{ij} = 1 \quad \text{for all } j \in J\,, \tag{3}$$

$$x_{ij} = 0 \text{ or } 1$$

In this formulation $I \equiv \{1, 2, ..., m\}$ is a set of agent indices, $J \equiv \{1, 2, ..., n\}$ is a set of task indices, $c_{ij}$ is the cost incurred if agent $i$ is assigned task $j$, $r_{ij}$ is the resource required by agent $i$ to do task $j$ and $b_i > 0$ is the amount of resource available to agent $i$. The natural interpretation of the decision variable is

$$x_{ij} = \begin{cases} 1 & \text{if agent } i \text{ is assigned task } j, \\ 0 & \text{otherwise}. \end{cases}$$

This problem takes on more of the appearance of the classical assignment problem when each constraint (2) is scaled by dividing both sides of the inequality by the right-hand side value to obtain

$$\sum_{j\in J} k_{ij}x_{ij} \leqslant 1\,, \tag{4}$$

where

$$k_{ij} = r_{ij}/b_i\,.$$

It is apparent that the classical assignment problem is a special case of the generalized assignment problem in which $k_{ij} = 1$ for all $i \in I, j \in J$ and $m = n$. The stipulation vector of ones in (4) and (3) suggests that the generalized assignment problem is a special case of the generalized trans-

portation problem [13] in the same way that the classical assignment problem is a special case of the pure transportation problem. Implicit enumeration algorithms that solve a series of generalized transportation problems have been used to solve the generalized assignment problem [1, 12].

The generalized assignment problem may be interpreted as a specialized transportation problem in which the amount demanded at each destination must be supplied by a single origin if the parameter $r_{ij}$ is constant for each $i$ (i.e., $r_{ij} = r_j$ for all $i \in I$). Algorithms for this special case of the generalized assignment problem have been developed by DeMaio and Roveda [4] and by Srinivasan and Thompson [14].

In Section 2 of this paper, a branch and bound algorithm is given for solving the generalized assignment problem. A numerical example is provided in Section 3, and computational results are presented in Section 4.

## 2. The branch and bound algorithm

Our branch and bound algorithm is basically a conventional one of the Dakin type [3] with the novel feature that the bound is calculated in part by solving binary knapsack problems, rather than using linear programming. This approach greatly exploits the structure of the problem and yields, in a direct way, an integer solution that tends to be feasible. The efficiency of the algorithm is based on the minimal effort required to solve binary knapsack problems [8, 11] coupled with a LIFO treatment of candidate problems.

The basic concept in the bounding procedure of the algorithm is best illustrated with reference to the initial relaxation of problem (P). The relaxation is a natural one in which tasks are assigned to the least costly agent without regard to the limitations on multiple assignments. This relaxation is

$$
\text{(PR)} \qquad
\begin{aligned}
&\text{minimize} && \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij}\,, \\
&\text{subject to} && \sum_{i \in I} x_{ij} = 1 \quad \text{for all } j \in J, \\
& && x_{ij} = 0 \text{ or } 1\,.
\end{aligned}
$$

An obvious solution to (PR) is found by determining for all $j \in J$, an index $i_j \in I$ such that $c_{i_j j} = \min_{i \in I}\{c_{ij}\}$ and setting $x_{i_j j} = 1$ and $x_{ij} = 0$ for

all $i \in I$, $i \neq i_j$. This yields the lower bound

$$Z = \sum_{j \in J} c_{i_j j} .$$

Each of the constraints (2) in (P) is then checked for feasibility using the solution to (PR). Except for trivial problems, some of these constraints will be violated, and this fact can be used to further refine the lower bound. For notational convenience let

$$J_i \equiv \{j: x_{ij} = 1, j \in J\}$$

and let

$$I' \equiv \{i: \sum_{j \in J_i} r_{ij} x_{ij} > b_i, \ i \in I\}.$$

The lower bound $Z$ can be increased by the sum of the values of the objective functions obtained from solving for each $i \in I'$ the problem

$$(\mathrm{PK}_i) \qquad \begin{aligned} &\text{minimize} && z_i = \sum_{j \in J_i} p_j y_{ij} , \\ &\text{subject to} && \sum_{j \in J_i} r_{ij} y_{ij} \geqslant d_i, \\ & && y_{ij} = 0 \text{ or } 1 , \end{aligned}$$

where

$$d_i = \sum_{j \in J_i} r_{ij} x_{ij} - b_i,$$

$$p_j = \min_{k \in I - \{i_j\}} \{c_{kj} - c_{i_j j}\} .$$

Each problem ($\mathrm{PK}_i$) is a binary knapsack problem the solution to which designates those tasks which must be reassigned from agent $i$ to another agent in order to satisfy the resource restriction on agent $i$. The parameter $p_j$ represents the minimum penalty that will be incurred if task $j$ is reassigned. The optimal solution to ($\mathrm{PK}_i$), denoted $y_{ij}^*$, indicates those reassignments that lead to a minimal increase, $z_i^*$, in the value of $Z$. Thus a revised lower bound for (P) is

$$\mathrm{LB} = Z + \sum_{i \in I'} z_i^* .$$

The solution $y_{ij}^*$ also may be used to construct a new solution which may be feasible for (P) and which has an objective function equal to the re-

vised lower bound. In particular, the variables $x_{ij}$ for which the corresponding $y^*_{ij} = 1$ should be set equal to 0, and a variable $x_{kj}$ whose associated coefficient $c_{kj}$ satisfies $p_j = c_{kj} - c_{ij}$ should be set equal to one.

Computing the lower bound for (P) in the manner just described can be viewed as a specific application of the concept of Lagrangean Relaxation, a general approach for constructing lower bounds in integer programming [6]. We shall now develop the bound in terms of Lagrangean relaxation because it makes the validity of the revised lower bound LB obvious.

The Lagrangean relaxation is obtained by associating a multiplier $\lambda_j$ with each of the constraints $\Sigma_{i\in I} x_{ij} = 1$ to produce the problem

$$(\mathrm{PR}_\lambda) \qquad \begin{aligned} &\text{minimize} && \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} + \sum_{j\in J}\lambda_j\Big(1 - \sum_{i\in I} x_{ij}\Big), \\ &\text{subject to} && \sum_{j\in J} r_{ij}x_{ij} \leqslant b_i \quad \text{for all } i \in I, \\ & && x_{ij} = 0 \text{ or } 1 . \end{aligned}$$

Note that an equivalent form of the objective function is

$$\sum_{j\in J}\lambda_j - \max\Big[\sum_{j\in J}\sum_{i\in I}(\lambda_j - c_{ij})x_{ij}\Big] .$$

Thus $(\mathrm{PR}_\lambda)$ separates into a series of binary knapsack problems, one for each $i \in I$.

The magnitude of the bound provided by $(\mathrm{PR}_\lambda)$ is clearly dependent upon the values of $\lambda_j$. A good choice for the $\lambda_j$ would be the values of the optimal dual multipliers of the constraints (3) in the continuous version of (P). However, considerable computational effort would be required to solve the continuous version of (P) by linear programming. Another suitable choice for the $\lambda_j$ would be the set of optimal dual multipliers for the bounded variable linear program,

$$(\mathrm{PR}_L) \qquad \begin{aligned} &\text{minimize} && \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij}, \\ &\text{subject to} && \sum_{i\in I} x_{ij} = 1 \quad \text{for all } j \in J, \\ & && 0 \leqslant x_{ij} \leqslant 1 . \end{aligned}$$

It is particularly important to note that each optimal dual multiplier $\lambda_j$ for $(\mathrm{PR}_L)$ lies anywhere in the range $c_{1j} \leqslant \lambda_j \leqslant c_{2j}$, where $c_{1j}$ and $c_{2j}$ are

respectively the smallest and second smallest value of $c_{ij}$ for all $i \in I$. Optimal $\lambda_j$ for $(PR_L)$ are, therefore, very easy to compute.

The principal result of this Lagrangean analysis is that the lower bound LB is identical to the bound provided by $(PR_\lambda)$ when each $\lambda_j$ is set equal to $c_{2j}$, the second smallest value of $c_{ij}$ for all $i \in I$. This assertion is easily verified if one substitutes $(1 - x_{ij})$ for $y_{ij}$ in each $(PK_i)$, observes that $p_j = c_{2j} - c_{1j}$ for all $j \in J$, and observes that those $x_{ij}$ which are equal to zero in the solution to (PR) can be set equal to zero in $(PR_\lambda)$. Thus, solving (PR) and subsequently solving $(PK_i)$ for all $i \in I'$ yields a bound equal to that provided by $(PR_\lambda)$ which is clearly a valid bound for (P).

The solution set for (P) is separated into two mutually exclusive and collectively exhaustive subsets based on the 0–1 dichotomy of variable values. The variable chosen to separate on, $x_{i^*j^*}$, is the one among those with $y^*_{ij} = 0$ in the optimal solution to the $(PK_i)$ for which

$$t_{i^*j^*} = \max \{p_j/(r_{ij}/(b_i - \Sigma_{j \in F_i} r_{ij} x_{ij}))\} ,$$

where $F_i$ denotes the set of tasks assigned to man $i$. The variable chosen by this rule represents a job $j$ that is best kept with agent $i$ considering both the penalty for switching the job and the resources available to the agent. Separation creates two new candidate problems whose solution sets differ only in the value assigned to a particular variable. The candidate problem in which the separation variable is fixed to one is the problem examined next.

It should be reemphasized that the solutions of the binary knapsack problems contribute to the fathoming of candidate problems in two ways. First, the objective function values refine the lower bound, and secondly the solutions indicate some reassignments of tasks to other agents that may lead to a feasible solution. Computational results indicate that if the resource constraints are not overly restrictive, then making these reassignments frequently produces a feasible solution.

As the branching progresses, particularly as the algorithm proceeds down the "one-branch", an additional refinement can be made in the procedure for solving (PR). In identifying the least costly and second least costly agent for a given task, some variables may be forced to zero because $r_{ij} > b_i - \Sigma_{j \in F_i} r_{ij} x_{ij}$, where $F_i$ designates those tasks assigned to agent $i$ in the current candidate problem. This in turn may increase the bound obtained from (PR) as well as the penalties $p_j$ in the objective function of the $(PK_i)$.

## 3. Numerical example

The objective function values ($c_{ij}$) and the coefficients ($r_{ij}$) of the multiple assignment constraints for a problem with three agents and five tasks are shown in Table 1 below. An asterisk (*) indicates that the associated agent-task pair is not permissible. In this example, the right-hand side values ($b_i$) of the multiple assignment constraints are 28 for all $i \in I$.

Initially (PR) is solved to obtain $x_{11} = 1$, $x_{12} = 1$, $x_{33} = 1$, $x_{34} = 1$, $x_{15} = 1$ and all other $x_{ij} = 0$. The lower bound on the optimal solution to (P) provided by this solution to (PR) is 92.

Because the solution to (PR) is not feasible to (P) the binary knapsack problems

$$\begin{aligned} &\text{minimize} && 35y_{11} + 35y_{15} + 7y_{12}, \\ &\text{subject to} && 12y_{11} + 18y_{15} + 19y_{12} \geqslant 21, \\ & && y_{1j} = 0 \text{ or } 1 \quad \text{for all } j; \end{aligned}$$

$$\begin{aligned} &\text{minimize} && 16y_{34} + 4y_{33}, \\ &\text{subject to} && 22y_{34} + 14y_{33} \geqslant 8, \\ & && y_{3j} = 0 \text{ or } 1 \quad \text{for all } j \end{aligned}$$

are solved, and the optimal solutions are $y^*_{11} = 0$, $y^*_{12} = 1$, $y^*_{15} = 1$ and $y^*_{34} = 0$, $y^*_{33} = 1$ respectively. The objective functions of these knapsacks give a combined penalty of 46 which added to the lower bound 92 gives a revised bound of 138.

The variable $x_{11}$ is selected as the separation variable, and the candidate problem with $x_{11} = 1$ is solved next. The solution to the relaxation of this candidate problem is $x_{11} = 1$, $x_{32} = 1$, $x_{33} = 1$, $x_{34} = 1$, $x_{25} = 1$ and all other $x_{ij} = 0$ with $Z = 134$. However this solution is not feasible for (P) and the knapsack problem

Table 1

| | | Tasks | | | | | | | Tasks | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | 2 | 3 | 4 | 5 | | | 1 | 2 | 3 | 4 | 5 |
| | 1 | 14 | 38 | * | 26 | 14 | | 1 | 12 | 19 | * | 11 | 18 |
| Agents | 2 | 49 | * | 20 | 46 | 49 | Agents | 2 | 6 | * | 11 | 15 | 18 |
| | 3 | * | 45 | 16 | 10 | * | | 3 | * | 10 | 14 | 22 | * |
| | | Objective function coefficients | | | | | | | Multiple assignment coefficients | | | | |

$$\begin{aligned} &\text{minimize} && +\infty\, y_{32} + 16y_{34} + 4y_{33} \\ &\text{subject to} && 10y_{32} + 22y_{34} + 14y_{33} \geqslant 18, \\ & && y_{3j} = 0 \text{ or } 1 \quad \text{for all } j \end{aligned}$$

is solved. The optimal solution is $y^*_{32} = 0, y^*_{34} = 1, y^*_{33} = 0$, and the lower bound of 134 can be revised to 150. Note that when task 4 is reassigned to agent 1 we obtain the feasible solution to (P), $x_{11} = 1, x_{32} = 1, x_{33} = 1$, $x_{14} = 1, x_{25} = 1$ with an objective function value of 150. Thus this candidate problem is fathomed, and the candidate problem with $x_{11} = 0$ is considered next.

The optimal solution to the relaxed candidate problem when $x_{11} = 0$ is $x_{21} = 1, x_{12} = 1, x_{33} = 1, x_{34} = 1, x_{15} = 1$ and all other $x_{ij} = 0$. The bound provided by the relaxation is 127. Again the solution to (PR) is not feasible to (P) and the knapsack problems

$$\begin{aligned} &\text{minimize} && 35y_{15} + 7y_{12}, \\ &\text{subject to} && 18y_{15} + 19y_{12} \geqslant 9, \\ & && y_{1j} = 0 \text{ or } 1\ ; \end{aligned}$$

$$\begin{aligned} &\text{minimize} && 16y_{34} + 4y_{33}, \\ &\text{subject to} && 22y_{34} + 14y_{33} \geqslant 8, \\ & && y_{3j} = 0 \text{ or } 1 \end{aligned}$$

are solved. The optimal solutions $y^*_{12} = 1, y^*_{15} = 0$ and $y^*_{33} = 1, y^*_{34} = 0$ increases the lower bound from 127 to 138. The variable $x_{15}$ is selected as the branching variable, and the process is repeated.

The complete branch and bound tree for this example is in Fig. 1. The unlabeled numbers beside each node indicate the order in which the nodes were examined, and LB designates the lower bound on the node.

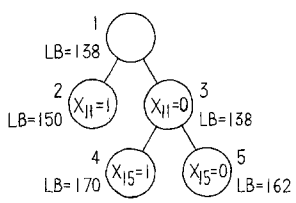


Fig. 1. Branch and bound tree for the example.

## 4. Computational results

We have programmed our algorithm and solved a variety of test problems. To demonstrate the efficiency of our approach relative to other algorithms, we have also solved subsets of our test problems with a general purpose integer programming code, an integer generalized network code and a special purpose code designed for a related class of problems.

The algorithm described in Section 2 has two characteristics which were exploited in implementing the algorithm. First, the LIFO treatment of candidate problems is beneficial in terms of both magnetic core storage and solution time requirements. With this arrangement only those restrictions associated with the current relaxation must be stored, and updating the candidate list requires very little effort. Empirical evidence suggests that the heuristic rule of examining the one branch first yields a good feasible solution early in the branching process. This in turn reduces the number of candidate problems that must be solved. The second important feature is that knapsack problems can be solved very rapidly in a relatively small amount of magnetic core, and good bounds are easily computed. Moreover, it has been suggested to the authors that some candidate problems might be fathomed using the bound provided by the continuous solution to the knapsack problems. This bound is extremely easy to compute and saves the time required to find an integer optimal solution to the knapsack problems.

Our program was written in FORTRAN IV and was tested on a CDC 6600 with 72 000 words of magnetic core available to the user. The program requires $4v + 7m + 10n + 5300$ words to solve a generalized assignment problem with $m$ agents, $n$ tasks and $v$ variables. The solution times reported in the tables below were measured by a real-time clock accurate to one millisecond and do not include time for input and output. The test problems include a number of randomly generated problems and one small problem which arose in an application. In the randomly generated problems, a uniform probability distribution was used to create $r_{ij}$ values between 5 and 25 and $c_{ij}$ values between 10 and 50. To establish binding multiple assignment constraints, each $b_i$ was set equal to $0.6 * (n/m) * 15 + 0.4 * R$, where $R = \max_{i \in I} \{\Sigma_{j \in J} r_{ij} x_{ij}\}$ and the $x_{ij}$ are the solution to the initial relaxation. In general, based on the range of values of the $r_{ij}$, one would expect random problems to be infeasible if each $b_i < (n/m) * 15$ and trivial if each $b_i \geqslant R$.

The data in Table 2 below provide a comparison of our algorithm

Table 2

| Ross and Soland | | | | RIP30C | | | | |
|---|---|---|---|---|---|---|---|---|
| Total solution time (sec) | Number of feasible solutions | Total number of nodes | Longest branch on tree | Number of knapsack problems | Total solution time (sec) | Number of feasible solutions | Number of iterations | Number of times L.P. called |
| 0.003 | 1 | 1 | 0 | 1 | 1.590 | 1 | 115 | 12 |
| 0.012 | 2 | 7 | 3 | 7 | 5.006 | 1 | 187 | 56 |
| 0.016 | 1 | 9 | 4 | 9 | 4.902 | 3 | 167 | 57 |
| 0.015 | 1 | 5 | 2 | 5 | 4.131 | 4 | 167 | 48 |
| 0.012 | 1 | 5 | 2 | 7 | 3.414 | 2 | 129 | 34 |
| 0.059 | 2 | 37 | 9 | 41 | 7.508 | 9 | 199 | 94 |
| 0.049 | 2 | 29 | 6 | 34 | 6.968 | 1 | 157 | 63 |
| 0.051 | 1 | 29 | 6 | 36 | 16.912 | 9 | 337 | 226 |
| 0.833 | 0 | 725 | 16 | 1084 | 3.260 | 0 | 101 | 45 |

Table 4

| $m$ | $n$ | Variables | Solution times | | | Number of feasible solutions | Total number of nodes | Longest branch on tree | Number of knapsack problems |
|---|---|---|---|---|---|---|---|---|---|
| | | | Median | Minimum | Maximum | | | | |
| 5 | 50 | 250 | 0.048 | 0.046 | 0.050 | 1 | 1 | 1 | 1 |
| 10 | 50 | 500 | 0.211 | 0.095 | 0.381 | 2 | 25 | 13 | 37 |
| 20 | 50 | 1000 | 0.333 | 0.199 | 1.568 | 1 | 19 | 10 | 45 |
| 5 | 100 | 500 | 0.093 | 0.079 | 1.026 | 1 | 1 | 1 | 2 |
| 10 | 100 | 1000 | 0.510 | 0.183 | 1.523 | 2 | 65 | 33 | 52 |
| 20 | 100 | 2000 | 0.630 | 0.411 | 1.068 | 1 | 35 | 18 | 56 |
| 5 | 200 | 1000 | 0.199 | 0.180 | 2.730 | 1 | 1 | 1 | 2 |
| 10 | 200 | 2000 | 1.951 | 0.362 | 2.760 | 1 | 165 | 83 | 216 |
| 20 | 200 | 4000 | 1.563 | 1.134 | 2.780 | 1 | 75 | 38 | 101 |

Table 3

| $m$ | $n$ | Variables | DeMaio and Roveda | Klingman and Stutz | Ross and Soland |
|---|---|---|---|---|---|
| 5 | 50 | 250 | 0.031 | 1.765 | 0.048 |
| 5 | 50 | 250 | 1.708 | 1.031 | 0.044 |
| 5 | 50 | 250 | TIME | 5.215 | 0.200 |
| 5 | 50 | 250 | 2.789 | 0.388 | 0.049 |
| 5 | 50 | 250 | 0.033 | 0.104[a] | 0.031 |
| 5 | 50 | 250 | 2.897 | 3.281 | 0.046 |
| 5 | 50 | 250 | 0.111 | 1.887 | 0.322 |
| 10 | 50 | 500 | 4.244 | 9.135 | 0.184 |
| 10 | 50 | 500 | 28.385 | 9.228 | 0.141 |
| 10 | 50 | 500 | 0.385 | 5.849 | 0.185 |
| 10 | 50 | 500 | 0.938 | 9.850 | 0.129 |
| 10 | 50 | 500 | 21.600 | TIME | 0.666 |
| 10 | 50 | 500 | 0.244 | 0.825 | 0.077 |
| 10 | 50 | 500 | TIME | TIME | 0.500 |
| 20 | 50 | 1000 | 8.139 | TIME | 0.292 |
| 20 | 50 | 1000 | CORE | TIME | 0.300 |
| 20 | 50 | 1000 | CORE | TIME | 0.372 |
| 20 | 50 | 1000 | 0.948 | 14.931 | 0.347 |
| 20 | 50 | 1000 | 10.941 | TIME | 0.508 |
| 20 | 50 | 1000 | 0.598 | 3.919 | 0.227 |
| 20 | 50 | 1000 | 0.032 | 1.451 | 0.207 |

[a] For this problem the optimal solution to the initial generalized network relaxation was integer valued.

against RIP30C [5], a general purpose integer programming code with an imbedded linear program for generating strong surrogate constraints. The application problem mentioned earlier in which $m = 5$, $n = 10$ and $v = 50$ was solved for nine progressively more restrictive sets of values for the $b_i$, the last set resulting in an infeasible problem. No randomly generated problems were solved using RIP30C because of magnetic core limitations.

The data in Table 3 provide a comparison of our algorithm against IPNETG and against a special purpose program developed by DeMaio and Roveda [4]. IPNETG is an experimental integer generalized network code recently developed by Klingman and Stutz [12]. IPNETG uses a branch and bound approach and solves generalized network problems to obtain the bounds. The algorithm of DeMaio and Roveda is designed to solve only the special case of the generalized assignment problem in which the parameter $r_{ij}$ is the same for each $i \in I$. The problems listed in Table 3 have this characteristic. The notation TIME indicates that a

code could not solve the problem in 50 seconds of central processor time, and the notation CORE indicates that a code required more than 72 000 words of magnetic core storage in order to solve the problem.

The data in Table 4 provide computational results for problems in which the parameter $r_{ij}$ varies for each $i \in I$. For each size problem in Table 4, seven randomly generated problems were solved. The median, minimum and maximum solution times are given, as well as the median values of the number of feasible solutions, nodes, knapsack problems, and length of the longest branch.

The computational results clearly establish the superiority of our algorithm for the generalized assignment problem. The solution times are relatively stable for each problem size in comparison to those of the other algorithms tested. As problem size increases, solution time increases but at a relatively slow rate. The data in Table 2 show that solution times tend to increase as the multiple assignment constraints become more restrictive, and that the algorithm is slow in determining that a problem has no feasible solution.

## Acknowledgment

We gratefully acknowledge the helpful comments of A.M. Geoffrion and Robert M. Nauss regarding both the Lagrangean relaxation interpretation of the algorithm and its computational implementation. We would also like to thank Professors DeMaio, Roveda, Klingman and Stutz for allowing us to use their programs.

## References

[1] V. Balachandran, "An integer generalized transportation model for optimal job assignment in computer networks", Working Paper 34-72-3, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pa. (November, 1972).

[2] A. Charnes, W.W. Cooper, D. Klingman and R. Niehaus, "Static and dynamic biased quadratic multi-attribute assignment models: solutions and equivalents", Center for Cybernetic Studies, Research Report CS 115, The University of Texas, Austin, Texas (January, 1973).

[3] R.J. Dakin, "A tree search algorithm for mixed integer programming problems", *Computer Journal* 8 (3) (1965) 250–255.

[4] A. DeMaio and C. Roveda, "An all zero–one algorithm for a certain class of transportation problems", *Operations Research* 19 (6) (1971) 1406–1418.

[5] A.M. Geoffrion, "An improved implicit enumeration approach for integer programming", *Operations Research* 17 (3) (1969) 437–454.

[6] A.M. Geoffrion, "Lagrangean relaxation for integer programming", *Mathematical Programming Study* 2 (1974) 82–114.

[7] A.M. Geoffrion and G.W. Graves, "Multicommodity distribution system design by benders decomposition", *Management Science* 20 (5) (1974) 822–844.

[8] H. Greenberg and R.L. Hegerich, "A branch search algorithm for the knapsack problem", *Management Science* 16 (5) (1970) 327–332.

[9] M.D. Grigoriadis, D.T. Tang and L.S. Woo, "Considerations in the optimal synthesis of some communication networks", Presented at the 45$^{th}$ Joint National Meeting of the Operations Research Society of America and The Institute of Management Sciences, Boston, Mass., April 22–24, 1974.

[10] D. Gross and C.E. Pinkus, "Optimal allocation of ships to yards for regular overhauls", Tech. Memorandum 63095, Institute for Management Science and Engineering, The George Washington University, Washington, D.C. (May, 1972).

[11] G.P. Ingargiola and J.F. Korsh, "Reduction algorithm for zero–one single knapsack problems", *Management Science* 20 (4) Part I (1973) 460–463.

[12] D. Klingman and J. Stutz, "Computational testing on an integer generalized network code", Presented at the 45$^{th}$ Joint National Meeting of the Operations Research Society of America and The Institute of Management Sciences, Boston, Mass., April 22–24, 1974.

[13] J.R. Lourie, "Topology and computation of the generalized transportation problem", *Management Science* 11 (1) (1964) 177–187.

[14] V. Srinivasan and G. Thompson, "An algorithm for assigning uses to sources in a speical class of transportation problems", *Operations Research* 21 (1) (1973) 284–295.