A New Heuristic for the Generalized Assignment Problem

Jaehun Joo*

ABSTRACT

The Generalized Assignment Problem(GAP) determines the minimum assignment of n tasks to m workstations such that each task is assigned to exactly one workstation, subject to the capacity of a workstation.

In this paper, we presented a new heuristic search algorithm for GAPs. Then we tested it on 4 different benchmark sample sets of random problems generated according to uniform distribution on a microcomputer.

1. Introduction

The Generalized Assignment Problem(GAP) determines the minimum assignment of n tasks to m workstations such that each task is assigned to exactly one workstation, subject to the capacity of a workstation. The GAP is different from the ordinary assignment problem since an amount of $a_{ij}(\geq 0)$ is required by a workstation i to do task j.

We may formulate the GAP as follows.

$$\text{Minimize } z = \Sigma_{i\in I} \Sigma_{j\in J} c_{ij}x_{ij} \qquad (1)$$

$$\text{s.t.} \quad \Sigma_{i\in I} x_{ij} = 1 \qquad \text{for each } j\in J \qquad (2)$$

$$\Sigma_{j\in J} a_{ij}x_{ij} \leq b_i \qquad \text{for each } i\in I \qquad (3)$$

$$x_{ij} = 0 \text{ or } 1 \qquad \text{for each } i\in I,\ j\in J \qquad (4)$$

$$I = \{1,\dots,m\}$$

$$J = \{1,\dots,n\}$$

Where, n : number of tasks

* Department of Information Industry, Dongguk University

m : number of workstations

a_{ij} : resource required by workstation i to do task j

c_{ij} : cost of assigning task j to workstation i

b_i : capacity of workstation i

x_{ij} : $\begin{cases} 1, & \text{if task j is assigned to workstation i} \\ 0, & \text{otherwise} \end{cases}$

There are many applications of the generalized assignment model which is an important class of network models. Fisher and Jaikumar[8] have developed a method for vehicle routing that is based on a generalized assignment model in which the tasks correspond to items to be delivered and the workstations to trucks. Ross and Soland[25] have shown certain facility location problems can be converted to the GAPs. Other applications include assigning software development tasks to programmers, assigning jobs to computer networks[6], and designing communication networks with capacity constraints[19]. Moreover, the problem appears as a subproblem in certain network design problems[20]. Recently Gavish and Pirkul[12] extend the GAP application to address the multi-resource GAP(MRGAP), in which workstations have a limited availability of a set of resources. The MRGAP has an important application in the trucking industry.

To find an optimal solution of the GAP, Ross and Soland[25] and Martello and Toth[22] developed a branch-and-bound method respectively. Fisher, Jaikumar and Van Wassenhove[9] developed an optimization algorithm exploiting branch-and-bound method in which a lower bound of the GAP is obtained from a Lagrangian relaxation in which capacity constraint set(2) is dualized. Guignard and Rosenwein[20] developed the dual based algorithm which improved the optimization algorithm of Fisher, Jaikumar and Wassenhove[9]. The dual based algorithm is so far known to be best among optimization algorithms. Amini and Racer[5] presented the heuristic approach which was called variable-depth-search heuristic(VDSH), to obtain the near optimum for the GAP.

The GAP is known to be NP-hard since the NP complete 2-partition problem is reducible to it. Given n real numbers $A_1, \ldots, A_n$, the 2-partition problem asks if there is a set $S \subseteq \{1, \ldots, n\}$ such that $\sum_{j \in S} A_j = \sum_{j \notin S} A_j$. This problem is equivalent to a GAP with m = 2, $a_{1j} = a_{2j} = A_j$ for all j, $b_1 = b_2 = \sum_{j \in N} A_j/2$ and c_{ij} arbitrary[9].

A heuristic approach may be justified to solve the large scale problems which might be represented by the GAP in the real-world. The VDSH of Amini and Racer has the limitation that may be unable to produce a feasible solution for some problems which are feasible.

In this paper, we present a new heuristic search algorithm based on the idea that the neighborhood of solutions historically found good must be thoroughly searched, whereas other regions should not be completely restricted for the search process. The heuristic consists of three phases. First phase scans a feasible solution by adjusting the available capacity. Second phase improves the incombent solution by exchanging the allocations of two tasks. Finally, the heuristic proceeds multiple swapping of task allocations. Then we tested our algorithm on 4 different benchmark sample sets of random problems generated according to uniform distribution on a microcomputer. To verify the efficiency of our algorithm more rigorously, we categorized each class of sample problems as "small"(n =10, 20, m = 3, 5) or "large"(n = 50, 100, 200, m = 5, 10, 20) and measured the computational running time and the relative error from optimum or lower bounds.

2. Heuristic Search Method

2.1 Outline of the Heuristic Search Method and Notations

To describe easily the heuristic, we represented the GAP as the table form which is similar to transportation table as shown in table 1. Each square(or cell) in ith row and jth column of table corresponds to a variable x_{ij}. Each cell (i, j) has a cost(c_{ij}) and a resource requirement(a_{ij}). Last column of table 1 includes capacities of workstations. We compute $\overline{a_{ij}}$ which is the criterion for making the initial assignment of tasks to workstations as follows:

$$\overline{a_{ij}} = c_{ij} * a_{ij} / b_i$$

Initial assignments based on the criterion of minimum $\overline{a_{ij}}$ are performed. When the further assignments by using this criterion are infeasible, the second phase of the heuristic performs two types of procedures to increase available capacity of workstations: movement of the task assigned to a workstation to a different workstation and swapping the resource assignments of two tasks. The second procedure of the phase attempts to assign an unassigned task to a workstation while simultaneously removing an assigned task from it.

If task j is assigned to workstation i, the cell of the table 1 corresponding to x_{ij} = 1 is called an assigned cell.

We defined the following value, variables, and sets:

b_i' : available capacity of workstation i

$$b_i' = b_i - \Sigma_{j\in J} a_{ij} x_{ij} \qquad \text{----------------------------------(5)}$$

$\bar{J}$: set of unassigned tasks(or columns):

$$\bar{J} = \{ j \in J: \Sigma_{i=1}^{m} x_{ij} = 0 \} \qquad \text{----------------------------------(6)}$$

AC_j : index of workstation to which task j is assigned.

S_i : set of tasks that may be moved from AC_j to workstation i

$S_i = \{j: a_{ij} \le b_i'$ and $AC_j \ne i$ for all $j \notin \bar{J}\}$ ------------------------------(7)

DC_i : set of tasks assigned to workstation i while satisfying

$a_{ij} \ge a_{ij*} - b_i'$ in case of assigning an unassigned task j^* to workstation i

$DC_i = \{j: AC_j = i$ and $a_{ij} \ge a_{ij*} - b_i'$ for all $j \notin \bar{J}$ in a given $j^* \in \bar{J}\}$----------(8)

S: set of pairs (i, j) satisfying S_i given by (7) for all I

$\triangle_{ij}$: amount of changes in cost by making a movement of the assignment of task j into workstation i

$\triangle_{ij} = c_{ij} - \delta_1 + \delta_2$ -----------------------------------(9)

where, $\delta_1 = \begin{cases} c_{i^0 j}, \text{ if } AC_j = i^0 \ne 0 \\ \text{Penalty\_cost, otherwise. Where penalty\_cost is big-M.} \end{cases}$

In case of $AC_j = 0$, the number of the unassigned column decreases.

$\delta_2 = \sum_{j \in J'} c_{i^0 j} - |J'|$*penalty_cost, where J' is a set of unassigned tasks that are able to be assigned to workstation i^0.

Before giving a further description of the heuristic method, we defined the following retangular loops associated with table 1. The rectangular loop is an ordered sequence of four different cells satisfying the following conditions:

1) Any two consequence cells lie in either the same row or same column
2) No three consecutive cells lie in the same row or column
3) The last cell in the sequence has a row or column in common with the first cell in the sequence.

In definition of a loop, the first cell is considered to follow the last cell, so the loop may be thought of as a closed path. The simple concept of the loop will be applied to improvements of a given solution in last phase of the heuristic.

Table 1: Table Form of the Generalized Assignment Problem

	1	...	j	...	n	b
1	$c_{11}(a_{11})$	...	$c_{1j}(a_{1j})$		$c_{1n}(a_{1n})$	b_1
.	.	.	.	.	.	.
i	$c_{i1}(a_{i1})$	.	$c_{ij}(a_{ij})$	.	$c_{in}(a_{in})$	b_i
.	.	.	.	.	.	.
m	$c_{m1}(a_{m1})$	.	$c_{mj}(a_{mj})$	.	$c_{mn}(a_{mn})$	b_m

2.2 Procedure

Phase Ⅰ: Initial Assignments

This phase makes assignments sequentially by using the criterion based on the minimum $\overline{a_{ij}}$.

Step 1: Compute $\overline{a_{ij}}$ for all i and j.

$\overline{a_{ij}} = c_{ij} * a_{ij} / b_i$

Let $\overline{J} = \{1, ..., n\}$, $AC_j = 0$ for all j, bi' = bi for all i, and $\overline{z} = 0$

Step 2: Determine a cell with minimum $\overline{a_{ij}}$ among the unassigned cells while satisfying constraint (3).

$(i^*, j^*) = \arg\min_{i \in I, j \in \overline{J}} \overline{a_{ij}}$ that satisfies $b_i' - a_{ij} \geq 0$.

Step 3: If we can't determine an incoming cell (i^*, j^*), go to step 4. Otherwise, let $\overline{J} = \overline{J} - \{j^*\}$, $AC_j = i^*$, $b_i' = b_i' - a_{i*j*}$, $\overline{z} = \overline{z} + c_{i*j*}$, and go to step 2.

Step 4: If $\overline{J} = \varnothing$, go to Phase Ⅲ. Otherwise, go to phase Ⅱ.

Phase Ⅱ: Adjusting Feasibility and Making Reassignments

In the case $\overline{J} \neq \varnothing$ through the initial assignments, the assignment criterion based on $\overline{a_{ij}}$ addressed to the cost minimization rather than feasibility. Thus we needed to relax the rigorous cost minimization. This step scans the unassigned cells on which an amount of resource requirement is less than the available capacity. Then this step tries to increase the overall feasibility that is the possibility of reducing $|\overline{J}|$.

Step 1: Determine S_i given by (7) for all i.

Step 2: If $S_i = \varnothing$ for all i, go to step 5. Otherwise, compute $\triangle_{ij}$ given by (9) for all (i,j)∈S, where S $= \cup_{i=1}^{m} S_i'$ and S_i' is a set of pairs (i', j') for all $j' \in S_{i'}$ in a given i'. Then select an incomming cell, $(i^*, j^*) = \arg\min_{(i,j) \in S} \triangle_{ij}$.

Step 3: Move the task j^* from AC_{j*} to i^*. Assign the unassigned task j to the workstation i^0 for all j∈ J', where $i^0 = AC_{j*}$ and J' is a set of unassigned tasks that are able to be assigned to the workstation i^0. Then update $\bar{z}$, AC_j, and $\bar{J}$. Also update b_i' and S_i for i = i^* and i^0.

Step 4: If $\bar{J} = \varnothing$, go to phase III. Otherwise, go to step 5.

Step 5: Select arbitrary an unassigned task $j' \in \bar{J}$ and compute WD_i given by (10) for all i. Determine a workstation i' in the given column j': $i' = \arg\min_i WD_i$. Then determine $DC_{i'}$ given by (8). In case of $DC_{i'} = \varnothing$, determine a workstation among the others excluding i' in the obvious analogous manner. Let a pair of i' and j' be an incomming cell. Determine an outgoing cell which is a pair of i' and j^0, where $j^0 = \arg\max_{j \in DC_{i'}} a_{i'j}$.

Step 6: In case of no incomming cell exists in step 5 and $S_i = \varnothing$ in step 2, the heuristic can't find a feasible solution. Otherwise, update $\bar{z}$, AC_j, and $\bar{J}$. Then update b_{i*}' and S_{i*}, and go to step 2.

In step 5, we can compute WD_i for $j' \in \bar{J}$ as follows:

$$WD_i = D_i * \text{NI} + \overline{c_{ij'}} * \text{NII for all i} \qquad \text{------------------------------(10)}$$

Where, $D_i = a_{ij'} - b_i'$

$\overline{c_{ij'}} = c_{ij'} - \min_i c_{ij'}$

NI and NII: weight given by user

The step 5 allows the user to choose the weight given to the feasibility and the cost. In the case of which NI is equal to 1.0 and NII is equal to 0.0, WD becomes D, where D means the diffrence between the available capacity and the resource requirement. The heuristic which gives the priority for the unassigned cell with the minimum value of D may become a nice method of quickly finding a feasible solution. In the case of which NI is equal to 0.0 and NII is equal to 1.0, WD is equal to $\bar{c}$, where $\bar{c}$ means the regret cost. Thus, we can find a good solution by using the criterion giving the priority of allocation to the unassigned cell with minimum regret cost.

In step 5, the problem determining the outgoing cell can be formulated into knapsack model[9]. But the heuristic chooses the cell having maximum a_{ij} as an outgoing cell.

Phase Ⅲ: Solution Improvements

To improve the incumbent solution, the heuristic performs two search processes. The first step either reassigns a task from one workstation to another, or swaps the assignments of two tasks. The second step simultaneously swaps the assignments of two or more tasks by applying a feasible loop construction method.

Phase Ⅲ-1: Single Swap

Step 1: If $S_i = \varnothing$ for all i, go to phase Ⅲ-2. Otherwise, go to step 2.
Step 2: Check cost savings accuring from the assignment of task j to workstation i for all (i, j) given by a pair of i and j$\in S_i$.
Step 3: Determine the best swap that results in largest cost savings.
Step 4: Update the incumbent solution and S_i.
Step 5: Repeat steps 1-4 until no further reduction in cost exists.

We need additional descriptions for steps 2 and 3 to understand them more easily. Let arbitrary pair of i and j in a given S_i be (i^*, j^*). Move a task j^* from workstation AC_j to i^* and compute the changed amounts of cost, $\delta_1 = c_{i*j*} - c_{i0j*}$, where $i^0 = AC_{j*}$. Let the set of tasks that satisfy the capacity constraint, $b_{i0} + a_{i0j*} \geq a_{i0j}$ for j $\neq j^*$ and $AC_j \neq i^0$ be J'. Then determine a task j^0 producing the largest decreases in cost among the elements of J' and put $\delta_2 = c_{i0j0} - c_{i'j0}$, where $i' = AC_{j0}$.
In step 3, we can compute δ as follows:

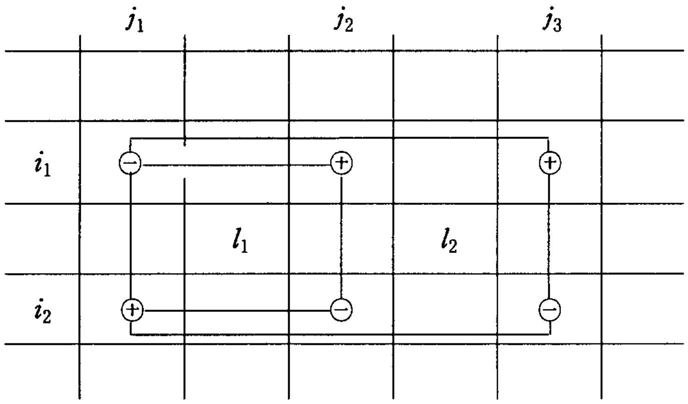
$$\delta = \begin{cases} \delta_1, \text{ if } \delta_2 \geq 0 \\ \\ \delta_1 + \delta_2, \text{ Otherwise} \end{cases}$$

In case of $\delta_2 \geq 0$, reassign the task j^* to workstation i^*. (i^*, j^*) is called an incomming cell and (i^0, j^*) is an outgoning cell. Otherwise, reassign the task j^* to workstation i^* and the task j^0 to i^0. (i^*, j^*) and (i^0, j^0) become the incomming cells, and (i^0, j^*) and (AC_{j0}, j^0) become the outgoing cells.

Phase Ⅲ-2: Multiple Swaps

To permute the assignments of p tasks, p≤n, this step constructs a rectangular loop composed of four cells at each iteration and finds a feasible loop with additional incomming and outgoing cells to

Table 2: An example of rectangular loop



improve the incumbent solution. The method permuting the assignments of p tasks is as follows.

Assumed that $AC_{j1} = i_1$, $AC_{j2} = i_2$, and $AC_{j3} = i_2$ in table 2. The loop l_1 that satisfies two constraints, $b'_{i1} + a_{i1j1} \geq a_{i1j2}$, and $b'_{i2} + a_{i2j2} \geq a_{i2j1}$ is feasible. If $b'_{i} + a_{i1j1} \geq a_{i1j3}$ and $b'_{i2} + a_{i2j3} \geq a_{i2j1}$, a loop l_2 is also feasible. In case of moving the task j_1 to a given workstation i_2, multiple feasible loops may exist. And also we may move the task j_1 from workstation i_1 to another workstation i, where $i \neq i_2$. Therefore many feasible loops may result from making movement of task j_1. Let the set of feasible loops accured by making movement of a task j_1 be L'.

In case of either $b'_{i1} + a_{i1j1} - a_{i1j2} \geq b'_{i1}$ or $b'_{i2} + a_{i2j2} - a_{i2j1} \geq b'_{i2}$ in a given feasible loop l_1, we may reduce the cost by moving the task j that is neither j_1 nor j_2, to i_1 or i_2. Let the largest decrease in cost through these movements be δ_2. The loop that produces the greatest cost savings δ among the elements of L' results in best reassignments of p tasks. Assumed that the best feasible loop is $(i_1, j_1) - (i_2, j_1) - (i_2, j_2) - (i_1, j_2)$. We can figure out δ as follows:

$$\delta = \begin{cases} \delta_1, \text{ if } \delta_1 < 0 \text{ and } \delta_2 \geq 0 \\ \\ \delta_1 + \delta_2, \text{ Otherwise, where } \delta_1 = c_{i1j2} + c_{i2j1} - c_{i1j1} - c_{i2j2} \end{cases}$$

To reduce redundant scan of loops, this phase proceeds the following steps.

Step 1: Put $j_1 = 1$

Step 2: Find the best permutation through movement of task j_1 by using loop construction and additional movements. Let an amount of change in cost be δ.

Step 3: If $\delta < 0$, reassign related tasks and update the incumbent solution.

Step 4: Put $j_1 = j_1 + 1$.

Step 5: Repeat steps 2-4 until $j_1 = n$ and go to step 6.

Step 6: In case of no further reduction in cost exists, terminate. Otherwise, go to step 1.

3. An Example

Let us illustrate the previous algorithm using a simple problem. This problem has four tasks and three workstations. Table 3 shows a table form of the generalized assignment problem including $\overline{a_{ij}}$.

Table 3: Table Form of the Example Problem

$c_{ij}(a_{ij})$ [$\overline{a_{ij}}$]				b_i
7(15) [5.5]	9(15) [7.1]	24(10) [12.6]	27(5) [7.1]	19
46(12) [42.4]	17(8) [10.4]	15(12) [13.8]	11(12) [10.1]	13
30(8) [13.3]	4(15) [3.3]	12(10) [6.6]	20(14) [15.5]	18

Phase I: Initial Assignment

As shown in table 4, assign 2nd task to 3rd workstation, 1st task to 1st workstation, and 4th task to 2nd workstation.

Table 4: Initial Assignments

c(a)				b'
7(15) *	9(15)	24(10)	27(5)	4
46(12)	17(8)	15(12)	11(12) *	1
30(8)	4(15) *	12(10)	20(14)	3

* indicates the assigned cell

AC = (1, 3, 0, 2)

$\overline{z}$ = 22+100(penalty-cost)

$\overline{J}$ = {3}

Phase Ⅱ: Adjusting Feasibility and Making Reassignment

$S_i = \varnothing$ for all i.

$j' = 3,$

D = (6, 11, 7)

$\bar{c} = (12, 3, 0)$, NI = NII = 1,

$WD = (18, 14, 7)$

$i' = \arg\min WD = 3$

Check DC_i for the incoming candidate cell (3, 3).

$DC_3 = \{2\}$

$j^0 = \arg\max_{j \in DC_i} a_{i'j} = 2$

Incomming cell (i', j'): (3, 3)

Outgoing cell (i', j^0): (3, 2)

Reassign tasks as shown in table 5.

$\bar{z} = 30+100$

$\bar{J} = \{2\}$

$b' = (4, 1, 8)$

Table 5: Solution Table after the Reassignments of Jobs

c(a)				b'
7(15) *	9(15)	24(10)	27(5)	4 ·
46(12)	17(8)	15(12)	11(12) *	1
30(8)	4(15)	12(10) *	20(14)	8

$S_3 = \{1\}$

$J' = \{2\}$

$\triangle_{31} = 30-7+9-100$(penalty-cost) $= 32-100$

Incomming cell $(i^*, j^*) = \arg\min_{(i,j) \in S} \triangle_{ij} = (3, 1)$

Outgoing cell $(i^0, j^*) = (1,1)$

Incomming cell $(i^0, j') = (1, 2)$

$\bar{z} = 62$

$b' = (4, 1, 0)$

$\bar{J} = \varnothing$

Table 6: Final Solution

c(a)				b'
7(15)	9(15) *	24(10)	27(5)	4
46(12)	17(8)	15(12)	11(12) *	1
30(8) *	4(15)	12(10) *	20(14)	0

AC = (3, 1, 3, 2)

The heuristic terminates the procedures with the optimum solution as shown in table 6.

Phase Ⅲ: Solution Improvements

To describe the phase III of the heuristic, we assumed that an initial solution is given as table 7.

b' = (4, 1, 3)

$\bar{J}$ = ∅

AC = (2, 3, 1, 1)

$\bar{z}$ = 101

Table 7: Initial Feasible Solution

c(a)				b'
7(15)	9(15)	24(10) *	27(5) *	4
46(12) *	17(8)	15(12)	11(12)	1
30(8)	4(15) *	12(10)	20(14)	3

Phase Ⅲ-1: Single Swap

S_i = ∅ for all i

Phase Ⅲ-2: Multiple Swaps

Table 8 shows a feasible loop and the additional movememt of task 3 for j_1 = 1.

Feasible loop: (2,1)-(3, 1)-(3, 2)-(2, 2)

Incomming and outgoing options through the additional movement: (3, 3), (1, 3)

$\delta_1 = -3$

$\delta_2 = -12$

$\delta = -15$

Incomming cells: (3, 1), (2, 2), (3, 3)

Outgoing cells: (2, 1), (3, 2), (1, 3)

AC = (3, 2, 3, 1)

b = (14, 5, 0)

$\bar{z} = 86$

Table 8: Best Feasible Loop Including Additional Move for $j_1 = 1$

c(a)				b'
7(15)	9(15)	24(10)⊖ *	27(5) *	4
46(12)⊖ *	17(8)⊕	15(12)	11(12)	1
30(8)⊕	4(15)⊖ *	12(10)⊕	20(14)	3

⊕ and ⊖ indicate the incomming cell and the outgoing cell, respectively.

Table 9 shows the updated solution and a feasible loop for $j_1 = 2$.

Feasible loop: (2, 2)-(1, 2)-(1, 4)-(2, 4)

Incomming and outgoing options through the additional movement: null

$\delta_1 = -24$

$\delta_2 = 0$

$\delta = -24$

Incomming cells: (1, 2), (2, 4)

Outgoing cells: (2, 2), (1, 4)

AC = (3, 1, 3, 2)

b = (4, 1, 0)

$\bar{z} = 62$

Table 9: Best Feasible Loop for $j_1 = 2$

c(a)				b'
7(15)	9(15)⊕	24(10)	27(5)⊖ *	14
46(12)	17(8)⊖ *	15(12)	11(12)⊕	5
30(8) *	4(15)	12(10) *	20(14)	0

Table 10: Final Solution

c(a)				b'
7(15)	9(15) *	24(10)	27(5)	4
46(12)	17(8)	15(12)	11(12) *	1
30(8) *	4(15)	12(10) *	20(14)	0

No further feasible loop exists. The final solution of the example problem is shown in table 8 and it is global optimum.

4. Computational Experiments

4.1 Computational Experiments

The algorithm was programmed in FORTRAN and executed on a microcomputer. The heuristic requires the following computer memory:

Storage requirements for input informations(a_{ij}, c_{ij}, and b_i): 2mn+m

Storage requirements for $\overline{a_{ij}}$ and b_i' : mn+m

Storage requirements for linked lists of $\overline{J}$ and AC_j : 2n

Storage requirements for S_{ij} and D_{ij} : 2mn

Storage requirements for building the loop and improving the solutions: 5n+4m

Thus, the heuristic requires a total of 5mn+7n+6m. On the other hand, Martello and Toth's HGAP, and Amini and Racer's VDSH requires 3mn+7n+6m and 4 n^2+6mn+5n+3m, respectively.

We tested it on a sample set of random problems generated from uniform distributions. This sample set consists of four classes which are referred to A, B, C, and D types.

Four different problems were generated from the following distributions:

A: a_{ij} and c_{ij} are integer from a uniform distribution between 5 and 25 and 1 and 40, respectively.
$b_i = 9(n/m) + 0.4 \max_{i\in I} \Sigma_{j\in J_i*} a_{ij}$, where
$J_i* = \{j\in J: i=\arg(\min_i c_{ij})\}$.

B: Same as A for a_{ij} and c_{ij}. b_i = 0.7 of b_i in A.

C: Same as A for a_{ij}. $b_i = 0.8\ \Sigma j \in J\ a_{ij}/m$.

D: Same as C for b. a_{ij} is integer from a uniform distribution between 1 and 100.

$c_{ij} = 111 - a_{ij} + K$. K is integer from a uniform distribution between -10 and 10.

The above four different types of problems are benchmarks which were used to test the algorithms developed by many researchers[5, 9, 20]. However, on the problem classes, A, B, and C, Fisher, et al., and Ross and Soland tested the problems with integer generated from a uniform distribution between 1 and 25 for c_{ij} while Amini and Racer generated values of c_{ij} between 1 and 40.

In this paper, to compare our heuristic method with the previous heuristic menthods, we categorized problems of each class as "small"(n = 10, 20, m = 3, 5) or "large"(n = 50, 100, 200, m = 5, 10, 20) according to Amini and Racer's classification. Since the GAP is NP-hard, the performance of optimization methods degrades severely with increased problem size. As a result, we compared the efficiency of our heuristic algorithm with that of optimization algorithm which was developed by Joo[1] for small test problems. The optimization algorithm represents the GAP as network for solving it. Then it scans the optimal path by using the fathoming rule which exploits the lower bound and backtracking.

However, to more adequately reflect real-world conditions, the large problems are included in measuring performance of the heuristic algorithm in terms of Central Processing Unit(CPU) running time and the relative errors from lower bounds. Two kinds of lower bounds are calculated. The first is computed by linear programming(LP) relaxation of the GAP. Hereafter, the lower bound is called LP. Since we could not obtain the optimal solutions on the large-size problems, we calculated the lower bound using LP relaxation in order to obtain the relative error. The lower bounds were calculated by the generalized network solver that was developed by Joo and Kim[2]. The other is computed by removing the capacity constraints(3). The lower bound is called LLB. It equals to the value that is calculated by allocating each task to any workstation subject to unlimited capacity. Hence errors of the heuristic are conservatively estimated in order to compare it with the previous heuristic methods.

Tables 11 and 12 show experimental results for "small" and "large" problems respectively. Each entry of tables 11 and 12 was obtained by randomly generating and solving ten different problems for each class type. Therefore, the algorithm was tested on a sample set of 520 random problems. The CPU time and relative errors are the average of results for ten problems. To obtain the ralative errors of the heuristic and ensure the consistency of experiment, we used the same random number seeds for all types of problems and all sizes of problems as follows:

No. of problem	Seed
1	12356.0
2	23456.0
3	65321.0
4	67890.0
5	99459.0
6	109880.0
7	109980.0
8	110990.0
9	145027.0
10	209135.0

All random numbers were generated by the subroutine of Microsoft Fortran optimizing compiler version 4.0.

Table 11: Average Solution CPU Time and Relative Error for "Small" Problems

Problem Class	Problem Size n	m	CPU Time (max)	Error from Optimum[1] (max)	Error from LP[2] (max)	Error From LLB[3]	CPU Time by Optimization
A	10	3	0.01(0.05)	0.000(0.000)	0.040(0.173)	0.080	0.02
	10	5	0.01(0.05)	0.000(0.000)	0.090(0.436)	0.168	0.04
	20	3	0.02(0.05)	0.001(0.020)	0.021(0.058)	0.034	0.04
	20	5	0.05(0.08)	0.000(0.000)	0.013(0.025)	0.049	0.02
B	10	3	0.01(0.05)	0.018(0.036)	0.170(0.417)	0.521	0.00
	10	5	0.01(0.06)	0.032(0.064)	0.280(0.706)	0.747	0.06
	20	3	0.01(0.06)	0.026(0.042)	0.116(0.241)	0.482	1.49
	20	5	0.03(0.06)	0.031(0.052)	0.144(0.323)	0.422	0.06
C	10	3	0.00(0.01)	0.007(0.034)	0.244(0.492)	0.539	0.00
	10	5	0.01(0.06)	0.031(0.052)	0.373(0.685)	1.211	0.06
	20	3	0.01(0.05)	0.028(0.051)	0.113(0.182)	0.349	0.06
	20	5	0.02(0.05)	0.029(0.053)	0.150(0.260)	0.588	71.40
D	10	3	0.02(0.05)	0.029(0.041)	0.084(0.140)	1.110	0.06
	10	5	0.02(0.05)	0.031(0.053)	0.123(0.192)	2.023	0.28
	20	3	0.00(0.04)	0.027(0.042)	0.041(0.062)	0.951	116.29
	20	5	0.04(0.06)	NA	0.059(0.068)	1.588	NA

1) $(\bar{z} - z^*)/z^*$, where $\bar{z}$ is objective value obtained by the heuristic. z^* is optimal value of the objective function obtained by optimization algorithm

2) $(\bar{z} - z_{LP})/z_{LP}$, where z_{LB} z_{LP} is lower bound of LP relaxation

3) $(\bar{z} - z_{LB})/z_{LB}$, where is lower bound calculated by removing capacity constraints

As shown in fourth column of table 11, the maximum average CPU time required to solve the problems is 0.05 second. Fifth column shows that the maximum average error from optimum is 3.2% on problem class B. For all small problem classes, the average errors from optimum and LP lower bound are 1.9 and 12.8 percent respectively. The heuristic algorithm exhibits very little sensitivity with respect to the problem size in terms of the relative errors on all problem classes.

The heuristic found the optimal solution on the problem class A. The problem class D is by far known to the most difficult problem type. On these small problems, the heuristic exhibits the relative error and CPU running time similar to those of problem classes B and C

Table 12 shows computational results of "large" problems. Third column of table 12 shows average CPU times. The maximum average CPU time of large problems is 357 seconds on 486 DX processor. An average error from LP lower bound on large problems is 0.059 which is 0.069 less than that of small problems. Figure 1 shows the relative errors from LP lower bound by problem class. As the number of jobs increases, the error decreases on each problem class. On the class A of small problems, the average relative error from LP is 4.1% and the heuristic finds optimal solutions. On the class A of large problems, the average relative error based on the LP lower bound is 0.5%. From this point of view, we may estimate that the heuristic finds the optimal solutions on the problem class A.

Since the computer codes of VDSH and HGAP are not available, the heuristic can't be directly compared with them. Thus we compared the performance of our algorithm with those of VDSH and HGAP through an indirect method. Even though test problems were produced by different random number generators using different seeds, empirical evidence indicates that the average relative error may reflect the performance of algorithm since each entry of table 12 is the average of results for ten problems and all parameters of each problem are generated from same range of distribution.

Fifth column of table 12 shows the relative errors from least lower bound for each algorithm. On thirty six cases, the average error rate of the heuristic(SGAP) is 19.1% less than that of HGAP. In particular, on the class problem D which is known to hard to solve, the heusristic achieved an average of 28.0% reduction in the relative error rate over HGAP. The VDSH could not find feasible solutions on 200 X 5 problem of class B, 100 X 5 and 200 X 5 problems of class C, and 50 X 20 problem of Class D. On thirty two cases excluding test problems not found feasible by the VDSH, the heuristic achieved an average of 2.8% reduction in the ralative error rate over VDSH. On the problem class D which consists of eight cases excluding one test problem not found feasible by the VDSH, the difference of errors between the heuristic and the VDSH is equal to 0.076.

Table 12: Average Solution CPU Time and Relative Error for "Large" Problems

Problem Class	Problem Size		CPU Time (max)	Error from LLB (VDSH, HGAP)1)	Error From LP (max)
	n	m			
A	50	5	0.62(0.82)	0.027(0.014, 0.021)	0.007(0.015)
		10	1.29(1.58)	0.020(0.026, 0.027)	0.008(0.018)
		20	2.99(3.57)	0.030(0.043, 0.045)	0.015(0.034)
	100	5	5.12(6.36)	0.003(0.004, 0.004)	0.0010.004)
		10	13.18(17.03)	0.010(0.007, 0.007)	0.005(0.007)
		20	30.44(41.40)	0.013(0.006, 0.006)	0.004(0.012)
	200	5	46.71(57.78)	0.002(0.002, 0.003)	0.001(0.001)
		10	114.90(146.86)	0.004(0.002, 0.003)	0.001(0.003)
		20	258.08(278.30)	0.008(0.002, 0.002)	0.002(0.004)
B	50	5	0.48(0.77)	0.251(0.334, 0.463)	0.064(0.115)
		10	1.00(1.43)	0.246(0.258, 0.399)	0.112(0.170)
		20	1.78(2.47)	0.223(0.275, 0.354)	0.141(0.231)
	100	5	2.58(4.07)	0.297(0.201, 0.329)	0.059(0.104)
		10	7.79(12.69)	0.197(0.205, 0.434)	0.074(0.123)
		20	22.87(33.07)	0.103(0.111, 0.205)	0.052(0.103)
	200	5	20.08(37.45)	0.244(NA2), 0.388)	0.010(0.036)
		10	74.08(87.67)	0.196(0.216, 0.498)	0.049(0.079)
		20	204.47(241.78)	0.179(0.100, 0.273)	0.081(0.135)
C	50	5	0.42(0.55)	0.349(0.349, 0.483)	0.068(0.093)
		10	0.71(1.26)	0.433(0.448, 0.708)	0.164(0.238)
		20	1.18(1.32)	0.790(0.924, 1.341)	0.391(0.480)
	100	5	3.47(5.00)	0.235(NA, 0.312)	0.056(0.100)
		10	9.21(13.02)	0.282(0.362, 0.647)	0.092(0.116)
		20	15.42(23.18)	0.387(0.506, 1.038)	0.168(0.283)
	200	5	24.16(30.43)	0.199(NA, 0.347)	0.031(0.083)
		10	69.45(87.67)	0.216(0.294, 0.546)	0.052(0.079)
		20	148.19(183.77)	0.335(0.280, 0.705)	0.098(0.150)
D	50	5	0.71(0.88)	1.487(1.511, 1.568)	0.034(0.052)
		10	1.59(1.98)	2.512(2.598, 3.747)	0.048(0.064)
		20	3.81(4.60)	3.675(NA, 3.758)	0.064(0.072)
	100	5	5.29(6.15)	1.493(1.518, 1.591)	0.025(0.029)
		10	16.35(19.66)	2.409(2.661, 2.813)	0.035(0.040)
		20	31.22(52.26)	3.740(3.834, 3.834)	0.050(0.062)
	200	5	47.94(72.28)	1.411(1.406, 1.539)	0.019(0.031)
		10	145.36(206.07)	2.470(2.517, 2.691)	0.025(0.030)
		20	357.56(417.27)	3.810(3.815, 4.040)	0.032(0.036)

1) The performance of the heuristic, VDSH developed by Amini and Racer(1994). The computational results of the HGAP which is developed by Martello and Toth are obtained from Amini and Racer.

2) VDSH couldn't find the feasible solutions in more than 5 out of 10 test problems

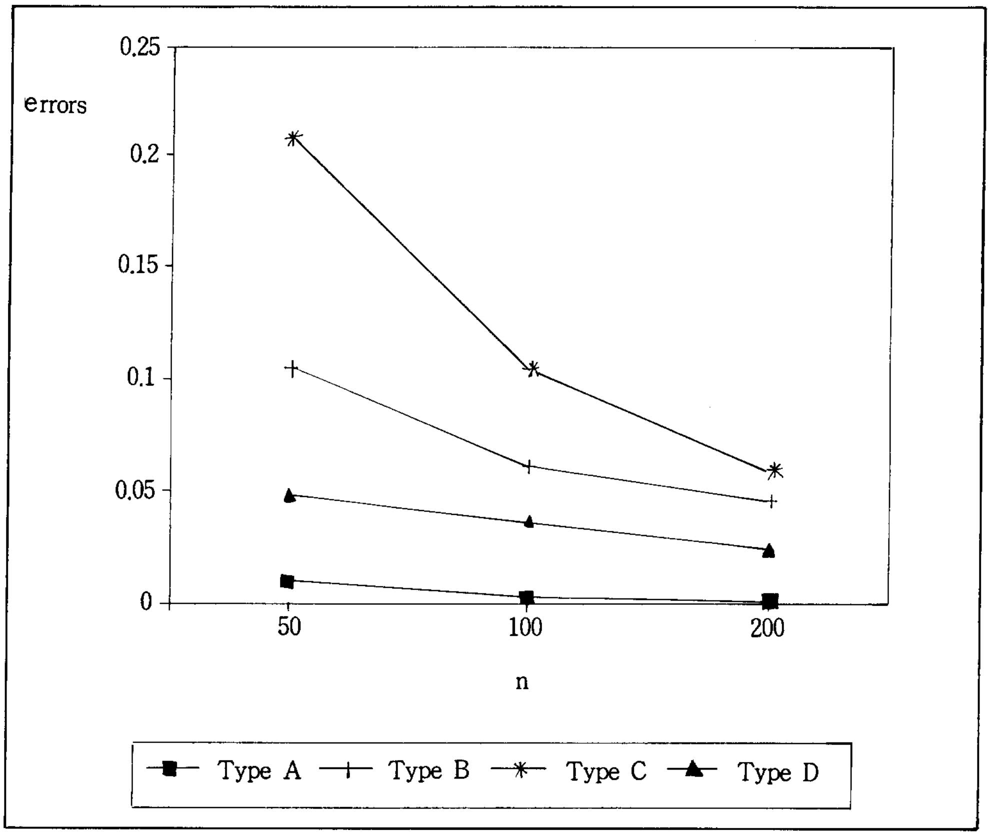


Figure 1 Relative Errors from LP lower bound

4.2 Statistical Analysis of Experimental Results

The efficiency of a heuristic may be measured by three criteria which are solution quality, CPU running time, and computer memory requirement. The advancement of computer technology makes the CPU time and the memory requirement less important.

On all test problems, we conducted the experiment on the microcomputer under MS-DOS while Amini and Racer did it on the VAX 6420 computer under the VAX/VMS operating systems. Thus, we can't compare the CPU running time under the situation where the computer codes of the previous heuristic methods are not available. As described in section 4.1, the heuristic(SGAP) and Martello and Toth's heuristic(HGAP) require O(mn) storage while Amini and Racer's heuristic(VDSH) requires O(n^2).

In this section, we focused on the comparison of the solution qualities obtained by alternative heuristic methods. If the relative error from the least lower bound(LLB) increases, the solution quality of a heuristic deteriorates.

The statistical analysis is designed to test null hypothesis as follows:

H1: On all problems, there is no significant difference between the heuristic(SGAP) and the previous heuristic methods(VDSH and HGAP).

H2: On each class problem, there is no significant difference between the heuristic and the previous heuristic methods.

To test hyothesises, T-test of SPSS/PC+ software is conducted. Table 13 shows means of relative errors for each class problem, which were obtained by alternative heuristic methods.

The null hypothesis(H1) associated with SGAP and HGAP is rejected at the 0.05 level of significance. However, the null hypothesis(H1) associated with SGAP and VDSH is not rejected. Thus, we conclude that the heuristic(SGAP), with an average error of 0.785, significantly outperforms the HGAP, with an average error of 0.976.

On each problem class except class A, the null hypothesis(H2) associated with SGAP and HGAP is rejected at the 0.05 level of significance. On the problem classes C and D, the null hypothesis(H2) associated with SGAP and VDSH is rejected at the 0.10 level of significance.

The statistical analysis indicates that the solution quality provided by our heuristic method is superior to that of HGAP. The heuristic(SGAP) is comparable to Amini and Racer's VDSH. In particular, on the problem classes C and D, the heuristic with the average errors of 0.398 and 2.416, outperforms the VDSH with the average errors of 0.451 and 2.482 with a 10% risk.

Table 13: Mean and P-value for relative errors

	SGAP	HGAP	P-Value	SGAP	VDSH#	P-Value
Class A	0.013	0.013	0.964	0.013	0.011	0.642
Class B	0.215	0.371	0.000**	0.211	0.212	0.964
Class C	0.358	0.679	0.001**	0.398	0.451	0.087*
Calss D	2.556	2.842	0.050**	2.416	2.482	0.060*
Overall	0.785	0.976	0.000**	0.747	0.775	0.021

Excludes test problems not found feasible by VDSH

** At the 0.05 level of significance, means of error between SGAP and HGAP are different.

* At the 0.10 level of significance, means of error between SGAP and HGAP are different.

5. Conclusion

The GAP is known to NP hard. According to Guignard and Rosenwein[20], 80 X 5 and 100 X 5 problems could not be solved with the existing core storage allocations on a DEC-10, although 40 X 10 and 50 X 10 problems were effectively solved. Their optimization algorithm is known to be most efficient.

We presented a new heuristic search algorithm for GAPs. A feasible solution is obtained by adjusting the available capacity of a workstation for the case of which there is at least one unassigned job existing after completion of the initial assignments which are conducted by using the criterion based on minimum cost per unit resource requirement. Finally, the heuristic performs the sophisticated procedures reassigning one or more job(s) to new workstation(s) to improve the incumbent solution.

We tested it on 520 random problems. An average error rate from optimimum is 1.9% on small problems of which the size is below 20 X 5. For large problems, we compared the heuristic with the previous heuristic methods which were developed by Amini and Racer, and Matello and Toth through the indirect method since their computer codes were not available. An average error from the lower bound which is computed by removing capacity constraints is 0.747 which is 0.028 less than over that of the VDSH that is known to most efficient heuristic method. The maximum average CPU running time of the heuristic is 357 seconds on microcomputer(486 DX processor), while that of the VDSH is 131.9 seconds on VAX 6420. Our heuristic method found the feasible solutions on all test problems while the VDSH couldn't find the feasible solutions on the 42 problems among 520 test problems. In particular, on the problem classes C and D, our heusristic method is superior to the VDSH at the 0.10 level of significance. But, we may not argue which algorithm is more efficient until both will be tested on the same problems and the same computer.

The statistical analysis indicates that our heuritic method is superior to the HGAP in terms of the solution quality. However, the limitation of the study lies in the fact that the heuristic couldn't be directly compared with the previous heuristic methods. (The computer code of SGAP is available from the author.)

REFERENCES

[1] Joo, J. H., "A New Algorithm Based on the Enumeration for the Generalized Assignment Problem," *Dongnam Journal of Management*, Vol. 10(1994), pp. 243-255.

[2] Joo, J. H. and K. Kim, "Improvements of Branch and Bound Algorithm for the Integer Generalized Network Problem," *Journal of the Korean Operations Research and Management Science Society*, Vol. 19, No. 2, 1994, pp. 1-19.

[3] Amini, M. M., "Network Reoptimization Algorithm: A Computational Comparison of Algorithmic Alternatives," *University of Microfilms*, International, Ann Arbor, MI(dissertation, Department of Operations Research, Southern Methodist University, Dallas, TX), Dec., 1989.

[4] ――――――, and R. S. Barr, "Network Reoptimization Algorithms: A Statistical Designed Comparison," *ORSA J. on Computing*, Vol. 5, No. 4, 1993, pp. 395-405.

[5] Amini, M. M. and M. Racer, "A Rigorous Computational Comparison of Alternative Solution Methods for the Generalized Assignment Problem," *Management Science*, Vol. 40, No. 7, July 1944, pp. 868-890.

[6] Balachandran, V., "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, Nov., 1972.

[7] Casco, D. O., B. L. Golden, and E.A. Wasil, "Vehicle Routing with Backhauls: Models, Algorithms, and Case Studies," *Vehicle Routing: Methods and Studies*, Elsevier Science Pub., 1988.

[8] Fisher, M. L. and R. Jaikumar, "A Generalized Assignment Heuristic for Vehicle Routing," *Networks*, Vol. 11, 1981, pp. 109-124.

[9] Fisher, M. L., R. Jaikumar, and L. N. Van Wassenhove, "A Multiplier Adjustment Method for the Generalized Assignment Problem," *Management Science*, Vol. 32, No. 9(1986), pp. 1095-1103.

[10] Forrest, J. J., J. P. H. Hirst and J. A. Tomlin, "Practical Solution of Large Mixed Integer Programming Problems with UMPIRE," *Management Science*, Vol. 20, No. 5(1974), pp. 736-773.

[11] Fox, B. L., "Data Structures and Computer Science Techniques in Operations Research," *Operations Research*, Vol. 26, No. 5(1978), pp. 686-717.

[12] Gavish, B. and H. Pirkul, "Algorithms for the Multi-Resource Generalized Assignement Problem," *Management Science*, Vol. 37, No. 6, 1991, pp. 695-713.

[13] Glover, F. "Tabu Search: A Tutorial," *Interfaces*, Vol. 20, No. 4, July-August, 1990, pp. 74-94.

[14] ――――, "Tabu Search - Part I," *ORSA J. on Computing*, Vol. 1, No. 3, 1989, pp. 190-206.

[15] ――――, "Tabu search - Part II," ――――――――――, Vol. 2, No. 1. 1990, pp. 4-34.

[16] Glover, F., J. Hultz, D. Klingman and J. Stutz, "Generalized Networks: A Fundamental Computer-Based Planning Tool," *Management Science*, Vol. 24, No. 12(1978), pp. 1209-1220.

[17] Golden, B. L. and W. R. Stewart, "Empirical Analysis of Heuristics," in E.L. Lawer et al.(Eds.), *The Traveling Salesman Problem*, John Wiley and Sons, New York, 1985.

[18] Greenberg, H. J., "Computational Testing: Why, How, and How Much," *ORSA J. on Computing*, Vol. 2, 1990, pp. 94-97.

[19] Grigoriadis, M. D., D. T. Tang and L. S. Woo, "Considerations in the Optimal Synthesis of Some Communication Networks," Presented at *the Joint National Meeting of ORSA/TIMS*, Boston,

MA, April, 1974.

[20] Guignard, M., M. B. Rosenwein, "An Improved Dual Based Algorithm for the Generalized Assignment Problem," *Operations Research*, Vol. 37, No. 4(1989), pp. 658-663.

[21] Hall, R. W., "Route Choice on Networks with Concave Costs and Exclusive Arcs," *Transportation Research*, 23b, 1989, pp. 103-121.

[22] Martello, S. and P. Toth, "An Algorithm for the Generalized Assignemnt Problem," in J. P. Brans(Ed.), *Operational Research* 81, North-Holland, Amsterdam, 1981, pp. 589-603.

[23] ―――――― and ―――, "Linear Assignment Problem," *Annals of Discrete Mathmatics*, Vol. 31, 1987, pp. 259-282.

[24] Papadimitriou, C. H. and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*, Prentice Hall, Englewood Cliffs, NJ, 1982.

[25] Ross, G. T. and R. M. Soland, "A Branch and Bound Algorithm for the Generalized Assignment Problem," *Math. Programming*, Vol. 8, 1977, pp. 92-103.

[26] Skorin-Kapov, J., "Tabu Search Applied to the Quadratic Assignment Problem," *ORSA J. on Computing*, Vol. 2, No. 1, 1990, pp. 33-44.