AN IMPROVED DUAL BASED ALGORITHM FOR THE GENERALIZED ASSIGNMENT PROBLEM

MONIQUE GUIGNARD
University of Pennsylvania, Philadelphia, Pennsylvania

MOSHE B. ROSENWEIN
AT&T Bell Laboratories, Holmdel, New Jersey

(Received April 1987; revision received February 1988; accepted July 1988)

The generalized assignment problem (GAP) determines the minimum cost assignment of n jobs to m agents such that each job is assigned to exactly one agent, subject to an agent's capacity. Existing solution algorithms have not solved problems with more than 100 decision variables. This paper designs an optimization algorithm for the GAP that effectively solves problems with up to 500 variables. Compared with existing procedures, this algorithm requires fewer enumeration nodes and shorter running times. Improved performance stems from: an enhanced Lagrangian dual ascent procedure that solves a Lagrangian dual at each enumeration node; adding a surrogate constraint to the Lagrangian relaxed model; and an elaborate branch-and-bound scheme. An empirical investigation of various problem structures, not considered in existing literature, is also presented.

The generalized assignment problem (GAP) determines the minimum cost assignment of n jobs to m agents such that each job is assigned to exactly one agent, subject to an agent's capacity. Applications of the GAP and existing algorithms are reviewed by Rosenwein (1986), and the problem appears as a subproblem in certain network design problems, e.g., Guignard and Rosenwein (1987). This paper designs and validates a dual based optimization procedure for the GAP which outperforms previous algorithms. Our algorithm yields a decrease in computational running times and in the number of enumeration nodes required to solve a GAP to optimality. Medium sized problems, unsolved in prior literature, are solved effectively. Although enumeration algorithms for the GAP exist, in particular, see Fisher, Jaikumar and Van Wassenhove (1986), they have not been demonstrated to solve problems with more than 100 decision variables. In addition, we determine which sizes and structures of problems are computationally the most difficult.

Section 1 describes the model, and Section 2 presents the improved enumeration algorithm. Section 3 summarizes computational results and benchmarks the algorithm with existing procedures. Some concluding remarks are given in Section 4.

1. THE GENERALIZED ASSIGNMENT PROBLEM

Let $I = \{i_1, \ldots, i_m\}$ be a set of agents, and let $J = \{j_1, \ldots, j_n\}$ be a set of tasks. For $i \in I, j \in J$, define c_{ij} as the cost of assigning task j to agent i, a_{ij} as the resource required by agent i to perform task j, and b_i as the resource availability of agent i. Also, y_{ij} is a 0-1 variable that is 1 if agent i performs job j and 0 otherwise. The model is given by (1)–(4).

Problem GAP

$$\text{Minimize} \sum_{i\in I} \sum_{j\in J} c_{ij}y_{ij} \tag{1}$$

subject to

$$\sum_{i\in I} y_{ij} = 1 \qquad \text{for each } j \in J \tag{2}$$

$$\sum_{j\in J} a_{ij}y_{ij} \leq b_i \qquad \text{for each } i \in I \tag{3}$$

$$y_{ij} = 0 \text{ or } 1 \qquad \text{for each } i \in I,\ \ j \in J. \tag{4}$$

The multiple choice constraints, (2), ensure that each task is assigned to exactly one agent. The knapsack constraints, (3), stipulate the capacity of each agent. Fisher et al. prove that the GAP is an NP-hard problem. The strong Lagrangian relaxation for the GAP is given by the following.

Subject classification Programming, integer: an algorithm for the GAP

0030-364X/89/3704-0658 $01.25
© 1989 Operations Research Society of America

Problem LR$_w$

$$\min_y \left\{ \sum_{i \in I} \sum_{j \in J} (c_{ij} - w_j) y_{ij} + \sum_{j \in J} w_j \mid (3), (4) \right\}.$$

The corresponding Lagrangian dual is given by $\max_{w \geq 0} v(\mathrm{LR}_w)$ where $v(\cdot)$ denotes the optimal value of problem $(\cdot)$. Also, define

$$Z = \min_y \left\{ \sum_{i \in I} \sum_{j \in J} (c_{ij} - w_j) y_{ij} \mid (3), (4) \right\}$$

$$W = \sum_{j \in J} w_j$$

$$p_j = \sum_{i \in I} y_{ij}, \quad \text{for each } j \in J$$

and

$$v(\mathrm{LR}_w) = Z + W.$$

The tree algorithm of Fisher et al. outperforms existing algorithms by an order of magnitude on tightly constrained problems with $m = 5$, $n = 20$. However, the authors do not report solving larger problems. The essential step of the Fisher-Jaikumar-Van Wassenhove (hereafter, FJV) algorithm is a Lagrangian dual ascent procedure that solves the Lagrangian dual at each enumeration node.

In this note, we propose certain extensions and enhancements to the FJV algorithm, and we deploy certain algorithmic methods not considered by the earlier procedure. These improvements lead to faster computational running times and shorter enumeration trees in solving GAPs to optimality. FJV maintains $p_j \leq 1$, for each j, throughout the algorithm. Although this condition is required for primal feasibility, it restricts the search over the w space in solving the Lagrangian dual. Addition of the surrogate constraint

$$\sum_{i \in I} \sum_{j \in J} y_{ij} \geq n \tag{5a}$$

or

$$\sum_{i \in I} \sum_{j \in J} y_{ij} \leq n \tag{5b}$$

to **LR**$_w$ also serves to potentially improve $v(\mathrm{LR}_w)$. The surrogate constraint is a convex combination of the dualized constraints, (2), and is required for primal feasibility. A complex branching scheme, combining depth-first and breadth-first branching, is used for enumeration. Finally, an effective implementation of subgradient optimization generates tighter bounds than the FJV algorithm at the root node with significantly reduced computational effort. Details of the enhancements and the overall enumeration algorithm are described in the following sections.

2. AN ENUMERATION ALGORITHM FOR THE GAP

Steps 1–4 are performed at each node of the enumeration algorithm. At the root node, subgradient optimization is used in place of Lagrangian dual ascent to solve the Lagrangian dual in Step 3.

Step 1. Initialize w.

Step 2. Compute Z and $v(\mathrm{LR}_w)$. Let y be an optimal solution of **LR**$_w$. If y violates (2), add a surrogate constraint (either 5a or 5b).

Step 3. Lagrangian dual ascent steps exploit violations of (2).

Step 4. Post-Lagrangian interchange heuristic attempts to resolve violations of (2) and tighten the upper bound.

Branch-and-bound is performed by branching on columns with p_j that is different from 1, and bounds are obtained by performing Steps 2–4 at each node. Execution terminates when all branches are fathomed.

2.1. Initialization of *w* and Addition of a Surrogate Constraint (Steps 1 and 2)

We initialize w_j at the root by setting $w_j = c_{ij} - \max_i \{c_{ij}\}$, for each $j \in J$. Thus, a job j may be assigned to none, one, or two or more agents, which allows for a search over a wide w space for the optimal Lagrangian dual. In contrast, Fisher et al. maintain $p_j \leq 1$, for each j, throughout their procedure. Empirically, improved bounds for tightly constrained problems are obtained if p_j is not restricted. At a lower level node, the multipliers are initialized by setting them equal to the final multiplier values obtained at the parent node.

The problem in y consists of m independent knapsack problems that are solved individually by the efficient knapsack code of Fayard and Plateau (1982). If a solution y to **LR**$_w$ violates $\sum_{i \in I} \sum_{j \in J} y_{ij} = n$, we add the "violated" part of the equation, i.e., (5a) or (5b), to the relaxation model as a tightening constraint. The addition of a surrogate constraint, generated from a model's relaxed constraints, potentially strengthens the Lagrangian bound. The FJV algorithm did not use surrogate constraints.

The Lagrangian problem with a surrogate constraint can be written as

$$\min_{i}\left\{\sum_{i\in I}\sum_{j\in J}(c_{ij}-w_j)y_{ij}\mid(3),(4),(5a)\text{ or }(5b)\right\}.$$

The surrogate constraint is, then, dualized into the objective function with multiplier θ. The strengthened relaxation is given by

$$\mathrm{LR}(\theta)=\min_{i}\left\{\sum_{i\in I}\sum_{j\in J}(c_{ij}-w_j+\theta)y_{ij}-\theta n\mid(3),(4)\right\}.$$

By not specifying a constraint on the sign of θ, the model is capable of accommodating a dualization of either (5a) or (5b). Since $\mathrm{LR}(\theta)$ is a piecewise linear and convex function of one variable, an optimal value of θ, θ^*, may be found with a search procedure, as proposed by Handler and Zang (1980) (see also Minoux 1975). At iteration k, let $\mathrm{LR}_k^+(\theta_k)$ (respectively, $\mathrm{LR}_k^-(\theta_k)$) be the equations of the rightmost (respectively, leftmost) segment of $\mathrm{LR}(\theta_k)$ with a positive (respectively, negative) slope and let θ_k be the abscissa of their intersection. If $\mathrm{LR}(\theta_k)=\mathrm{LR}_k^+(\theta_k)=\mathrm{LR}_k^-(\theta_k)$, $\theta^*=\theta_k$. Otherwise, let $k=k+1$ and repeat. The procedure iterates until the termination criterion is reached. No attempt is made to close a duality gap, if one exists.

2.2. Lower and Upper Bound Calculation (Steps 3 and 4)

A lower bound at each node, save the root, is computed by a Lagrangian dual ascent procedure that exploits violations of the dualized constraints to improve the Lagrangian bound. If a job j exists with p_j different from 1, i.e., if (2) is violated, the corresponding w_j may be adjusted upward (downward) to effectively increase (decrease) the attractiveness of including (excluding) this job in (from) exactly one knapsack solution. If a certain job j^* is unassigned, an increase in w_{j^*} results in an increase in W while Z remains constant and $v(\mathrm{LR}_u)$ increases. At the point where j^* is assigned to exactly one agent, Z is no longer constant. In fact, Z begins to decrease at the same rate that W increases and no bound improvement occurs. Similarly, if a job j^* is assigned to two or more agents, a decrease in w_{j^*} results in a decrease in W that is offset by a larger increase in Z. If W decreases by an amount Δ_{j^*}, Z increases by an amount $p_{j^*}\Delta_{j^*}$, $p_{j^*}>1$. The overall effect of an adjustment of w_{j^*} is to increase $v(\mathrm{LR}_u)$ by an amount $\Delta_{j^*}(p_{j^*}-1)$. At the point where j^* is assigned to exactly one agent, the increase in Z is offset by the decrease in W, and no bound improvement is possible. The Lagrangian dual ascent steps are described in greater detail in Guignard and Rosenwein (1989).

Calculation of the bound at the root of the enumeration tree by subgradient optimization improves the enumeration algorithm's performance. We implemented the subgradient procedure as follows. Let LB and UB denote the lower and upper bounds, respectively. Let k be an iteration counter, and λ_k is a parameter adjusted by the algorithm. At Step k, a lower bound LB_k is computed by $\mathrm{LB}_k=W+Z$. $\mathbf{LR}_u$ is solved at every iteration by computing the multipliers w_j^k, as

$$w_j^k=w_j^{k-1}-t_k\left(1-\sum_{i\in I}y_{ij}^k\right)$$

where

$$t_k=\lambda_k(\mathrm{UB}-\mathrm{LB}_k)\Big/\sum_{j\in J}\left(1-\sum_{i\in I}y_{ij}^k\right)^2$$

and y_{ij}^k is the value of y_{ij} at the kth iteration. If $\mathrm{LB}_k\leqslant\mathrm{LB}_{k-1}$, we set $\lambda_k=\lambda_{k-1}/2$. If $\lambda_k\leqslant\epsilon$, the subgradient algorithm terminates. In this implementation of the subgradient procedure, $\lambda_0=2$ and $\epsilon=0.001$. The subgradient procedure typically terminates in approximately 30 iterations at the root node. The relatively rapid termination is due to our method for updating λ_k.

If the subgradient procedure is used to solve the Lagrangian dual at each tree node, the enumeration algorithm converges slowly to optimality, as monotonic bound improvement is not guaranteed by the subgradient method. For certain problems, the number of nodes required to prove optimality exceeded the core storage. Furthermore, the number of subgradient iterations greatly exceeds the number of ascent steps at each level of the tree, except the root. Therefore, we opted for a bounding strategy that uses the subgradient method to solve the Lagrangian dual at the root and Lagrangian dual ascent at all other nodes.

Finally, we developed an interchange routine in an attempt to attain a feasible solution if the ascent procedure terminates with (2) violated. More importantly, this routine potentially can tighten the upper bound and, thereby, enhance the overall enumeration procedure. It attempts to assign an unassigned j to some knapsack i while simultaneously removing a multiply-assigned j from the same i. If a feasible solution is found, it is compared with the current upper bound for possible updating.

2.3. Branch-and-Bound

Steps 1–4 are performed at each node. A node is fathomed if its lower bound exceeds or is equal to the current upper bound. If a feasible solution value at a node is less than the current upper bound, the upper bound is updated. The procedure terminates if all pending nodes are fathomed.

If a j exists with p_j different from 1 following termination of ascent at a node, the enumeration algorithm must partition the node's solution set. The addition of (5b) ensures that at least one p_j is greater than 1. We select one j^* with p_{j^*} greater than 1 and create p_{j^*} branches at the node. For each branch, a distinct y_{ij} with $y_{ij^*} = 1$ in the current solution is fixed at 0. At each node, we branch on a j that corresponds to $j^* = \min_j \max_i \{a_{ij} \mid p_j > 1 \text{ and } y_{ij} = 1\}$ and initially explore the branch corresponding to $\max_i \{a_{ij^*}\}$. The variable y_{ij^*} is disqualified from solutions at descendant nodes. The branching rule minimizes the largest increase in slack in the generated candidate subproblems, i.e., fewer feasible solutions are likely to exist.

We initially use depth-first branching to explore the tree. If the duality gap at a pending node is no more than 1%, depth-first branching is temporarily abandoned. The Lagrangian dual ascent procedure, typically, yields increasingly smaller improvements in the lower bound as branching proceeds down the tree. If the duality gap at a pending node is no more than 1%, we backtrack to the parent node and proceed downward upon an unexplored branch. If an improved feasible solution is found, the algorithm may fathom dangling nodes, i.e., nodes that were not fathomed but whose bounds were within 1% of a previously incumbent feasible solution. If continued backtracking does not result in an improved feasible solution, the dangling, unfathomed nodes are fathomed by closing the duality gap through a resumption of depth-first branching. Our branching rules enhance the FJV branching scheme, which is a pure depth-first tree search.

Branch-and-bound is implemented with data structures that enable a unique path to be traced from the root to any pending node. A list of multiplier adjustments that occur at a node may be saved since, with the exception of the root node, the number of adjustments is small compared with the number of multipliers. Thus, the enumeration algorithm may reposition itself at any nonfathomed node and resume a tree search from that node. If a node is fathomed, storage space allocated to the node is freed and may be reused.

3. COMPUTATIONAL EXPERIENCE

The algorithm was programmed in FORTRAN and executed on a DEC-10. We tested it on a sample set of random problems, generated according to distributions A, B, C, and D described in Martello and Toth (1981). The letters A, B, C, and D refer to four different problem generations and are given by the following distributions.

A. a_{ij} and c_{ij} are integer from a uniform distribution between 5 and 25 and between 10 and 25, respectively. $b_i = 9(n/m) + 0.4 \max_{i \in I} \sum_{j \in J_i^*} a_{ij}$ where $J_i^* = \{j \in J \mid i = \arg(\min_i c_{ij})\}$.

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} and c_{ij}. $b_i = 0.8 \sum_{j \in J} a_{ij}/m$.

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.

Table I benchmarks the proposed algorithm. Each entry was obtained by randomly generating and solving ten different problems for each data class with $m = 5$, $n = 20$. Problems of dimension 5×20 represent the largest problems solved by the FJV algorithm. For results described in Tables I and II, the root bound was computed by our Lagrangian dual ascent procedure.

Table I illustrates that our algorithm compares favorably in the number of nodes and the average and maximum CPU running times required to solve tightly constrained problems. We also consider the effectiveness of certain enhancements in calculating the root bound. Since the FJV algorithm was not available, we recoded it according to its specifications

Table I
Algorithm Benchmark ($m = 5$, $n = 20$)

Data Set	Average Number of Nodes		Average (max) CPU Running Times			
	Fisher et al.	Guignard-Rosenwein	Fisher et al.		Guignard-Rosenwein	
A	2	2	0.645	(0.910)	1.325	(2.366)
B	28	11	14.930	(40.730)	5.039	(11.884)
C	25	20	13.750	(43.490)	8.316	(20.058)
D	126	84	67.606	(115.710)	67.337	(92.839)

(see Fisher et al.). For the ten C 5 × 20 problems, the root bound improved on average by 3.74% if the surrogate constraint was added and p_j was maintained $\leqslant 1$, and by 4.53% if the surrogate constraint was added and p_j was allowed to assume values > 1. In one extreme case, the bound increased by 10.33% using our initialization of w_j and the addition of (5a) or (5b). For all ten problems, our root bound was superior to the root bound computed by our implementation of the FJV algorithm. Addition of the surrogate constraint impacts the quality of the root bound more significantly than initialization schemes. The root bound was typically between 1–3% of the optimal solution.

A summary of the results appears in Table II. No performance benchmarking is possible for larger problems, for they are unsolved by previous algorithms. Ten problems were solved from each problem class with the exception of $m = 10$, $n = 40$ and $m = 10$, $n = 50$ for which only five problems were generated due to computer budget considerations. For B and C problems with mn larger than 500 and for D problems with mn larger than 150, core storage required to maintain the tree search exceeded allocated storage capacity. Type A problems were not solved for mn larger than 250 because their structure is not interesting i.e., (3) is not tight. The effect of the algorithm's performance of varying m and n was investigated. If mn is constant, but the ratio n/m increases, the algorithm's performance deteriorates. Larger problems, in fact, could not be effectively solved if n/m was too large, e.g., if $n/m > 10$. For instance, although 10 × 40 and 10 × 50 problems are effectively solved, 5 × 80 and 5 × 100 problems cannot be solved with the existing core storage allocation.

The performance of the enumeration algorithm is improved further if subgradient optimization is used to solve the Lagrangian dual at the root node. The root bound is improved, on average, by 1.36% for the ten C 5 × 20 problems and by 0.62% for the ten D 5 × 20 problems. Computational running time to calculate the bound is also reduced, e.g., by a factor of 20 for the D problems. Overall improvement to the FJV enumeration algorithm may be viewed as a two-step progression, as illustrated by Table III. HYBRID computes root bounds with subgradient optimization and lower level bounds with a Lagrangian dual ascent procedure. LDA computes each lower bound in the

Table II
Summary of Results

Data Set	m	n	No. of Problems Solved	Avg. No. of Nodes	Avg. CPU Seconds	Max. CPU Seconds
A	5	20	10	2	1.325	2.366
A	5	50	10	7	11.382	19.427
B	5	20	10	11	5.039	7.024
B	5	30	10	16	11.884	23.958
B	5	40	10	26	32.190	78.294
B	10	25	10	15	12.709	34.615
B	7	35	10	28	26.732	37.899
B	5	50	10	37	51.205	88.737
B	10	40	5	66	77.464	186.502
B	10	50	5	101	199.562	339.988
C	5	20	10	20	8.316	20.058
C	5	30	10	28	18.883	32.056
C	5	40	10	39	37.694	69.692
C	10	25	10	16	14.655	43.442
C	7	35	10	35	37.141	69.810
C	5	50	10	53	59.468	94.700
C	10	40	5	86	114.058	180.928
C	10	50	4[a]	156	268.434	308.402
D	5	20	10	84	67.336	92.839
D	5	30	5	102	168.240	273.655

[a] One other problem was generated but not solved because the number of nodes exceeded allotted core storage.

Table III
Performance Improvements by the HYBRID Algorithm ($m = 5$, $n = 20$)

Data	Avg. No. of Nodes			Avg. CPU Seconds		
Set	FJV	LDA	HYBRID	FJV	LDA	HYBRID
C	25	20	16	13.8	8.3	7.9
D	126	84	69	67.6	67.3	46.6

tree with a Lagrangian dual ascent procedure. Table III displays evidence that both HYBRID and LDA improve upon the FJV algorithm, for each requires fewer enumeration nodes and CPU seconds, on average, to solve GAPs to optimality. HYBRID outperforms LDA because at the root the number of Lagrangian dual ascent steps greatly exceeds the number of subgradient iterations, and furthermore, the subgradient yields improved bounds. However, Lagrangian dual ascent is a preferred method for solving the Lagrangian dual at all other nodes.

4. CONCLUSION

Our dual based algorithm for the GAP features: 1) an improved Lagrangian dual ascent procedure to solve the Lagrangian dual; 2) a complex branching strategy that shortens the tree search; 3) the addition of a surrogate constraint to the relaxed GAP; and 4) the use of subgradient optimization to compute bounds at the root node. Empirical testing reveals that our algorithm requires fewer enumeration nodes and CPU seconds than existing algorithms to solve the GAP to optimality. The algorithm also solves medium sized problems, unsolved by existing algorithms. (The computer code is available from the second author.)

ACKNOWLEDGMENT

This research was partially supported by the National Science Foundation under grant ECS-8508142.

REFERENCES

FAYARD, D., AND G. PLATEAU. 1982. An Algorithm for the Solution of the 0-1 Knapsack Problem. *Computing* **28,** 269–287.

FISHER, M., R. JAIKUMAR AND L. VAN WASSENHOVE. 1986. A Multiplier Adjustment Method for the Generalized Assignment Problem. *Mgmt. Sci.* **32,** 1095–1103.

GUIGNARD, M., AND M. ROSENWEIN. 1987. An Application of Lagrangean Decomposition to the Resource-Constrainted Minimum-Weighted Arborescence Problem. Working Paper 86-06-10, Department of Decision Sciences, University of Pennsylvania, Philadelphia.

GUIGNARD, M., AND M. ROSENWEIN. 1989. An Application-Oriented Guide to Lagrangean Dual Ascent Algorithms. *Eur. J. Opnl. Res.* (to appear).

HANDLER, G., AND I. ZANG. 1980. A Dual Algorithm for the Constrained Shortest Path Problem. *Networks* **10,** 293–310.

JÖRNSTEN, K., AND M. NÄSBERG. 1986. A New Lagrangian Relaxation Approach to the Generalized Assignment Problems. *Eur. J. Opnl. Res.* **27,** 313–323.

MARTELLO, S., AND P. TOTH. 1981. An Algorithm for the Generalized Assignment Problem. In *Proceedings of the 9th IFORS Conference,* Hamburg, Germany.

MINOUX, M. 1975. Plus Courts Chemins avec Contraintes: Algorithmes et Applications. *Ann. Télécommun.* **30,** 383–394.

ROSENWEIN, M. 1986. Design and Application of Solution Methodologies to Optimize Problems in Transportation Logistics. Ph.D. Dissertation, Department of Decision Sciences, University of Pennsylvania, Philadelphia.