A LINEAR RELAXATION HEURISTIC FOR THE GENERALIZED ASSIGNMENT PROBLEM

Michael A. Trick *

April, 1989; Revised March, 1991

Abstract

We examine the basis structure of the linear relaxation of the generalized assignment problem. The basis gives a surprising amount of information. This leads to a very simple heuristic that uses only generalized network optimization codes. Lower bounds can be generated by cut generation, where the violated inequalities are found directly from the relaxation basis. An improvement heuristic with the same flavor is also presented.

1 Introduction

The generalized assignment problem is to assign jobs to machines, where each machine has a capacity and each job has a size and a cost, each possibly dependent on the machine to which it is assigned. This problem has applications in vehicle routing ([4, 9]), distribution systems ([1]), facility location ([17]), and other fields in operations research.

*Supported in part by postdoctorate fellowships from the Institut für Ökonometrie und Operations Research, Bonn, West Germany and NATO, awarded by NSERC, Canada. Address: Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA, USA, 15213. The author would like to thank Annelie von Arnim for many helpful discussions.

The generalized assignment problem is NP–complete ([5]) so it is unlikely that any efficient solution method will be found. Previous work on the generalized assignment problem has concentrated on exact solutions to the problem using enumerative schemes with bounding methods (Martello and Toth [12], Ross and Soland [16], and Fisher, Jaikumar and Van Wassenhove [5]). Often, exact solutions are not necessary, and the time to find them may be prohibitive. Instead, good solutions are required quickly. Benders and van Nunen ([1]) examine the linear relaxation of the generalized assignment problem and show that the number of jobs not uniquely assigned to one machine is quite small and suggest that the "solutions of the relaxed problem will be a good starting point for a heuristic that assigns the remaining split jobs."

In this paper, we continue the examination of the basis structure of the linear relaxation and show that more information is available. This leads to an alternative proof of the theorem of Benders and van Nunen, a heuristic for the generalized assignment problem, constraint generation techniques for generating lower bounds, and an improvement heuristic for finding better solutions. These methods use extremely fast generalized network subroutines and can therefore solve very large problems quickly.

2 Formulation and Heuristics

Consider the problem of assigning n jobs to m machines. Each machine i has a capacity b_i; each job j has a size a_{ij} and a cost c_{ij} if it is assigned to machine i. Define a 0–1 variable x_{ij} to be 1 if job j is assigned to machine i and 0 otherwise. The *generalized assignment problem (GAP)* is to

Minimize $\sum_{i,j} c_{ij} x_{ij}$
Subject to

$$\sum_j a_{ij} x_{ij} \leq b_i, \quad i \in \{1, \ldots, m\} \tag{1}$$

$$\sum_i x_{ij} = 1, \quad j \in \{1, \ldots, n\} \tag{2}$$

$$x_{ij} \in \{0, 1\}, \quad i \in \{1, \ldots, m\}, \ j \in \{1, \ldots, n\}. \tag{3}$$

The *linear relaxation* of GAP *(LGAP)* replaces the integrality constraints

(3) with non–negativity constraints. Since LGAP is a special generalized network (network with multipliers), the basis structure of LGAP is well known (see particularly Brown and McBride [2]). The variables in any basis can be represented by a set of one–trees (trees with one extra edge, possibly a slack edge, forming a single cycle in each component). The nodes of these one–trees come in two types: *job nodes* and *machine nodes*, with one for each job and machine respectively. A variable x_{ij} in the basis corresponds to an edge between job node j and machine node i.

We can specialize the basis structure in [2] for LGAP. First, since the underlying problem is bipartite, so is any basis graph. Second, the basis cycles have special structure, outlined in the following lemma. We will state this lemma assuming that there are artificial variables associated with each job. If a solution is infeasible so that some job is not completely assigned, then the corresponding artificial variable is in the basis.

Lemma: *Any basic solution to LGAP consists of a number of connected components, where each component has at most one of:*

1. *A machine not used to capacity by the basic solution,*
2. *A job not scheduled completely by the basic solution, and*
3. *A cycle of alternating job and machine nodes, where each machine is used to capacity by the solution and each job is scheduled completely by the solution.*

PROOF: This follows directly from the fact that there is exactly one cycle in each component, and these are the three possibilities for a cycle. □

For any solution to LGAP, define a *split job* to be a job j associated with at least one fractional value x_{ij}. The following theorem is proved by Benders and van Nunen [1]. This alternative proof stresses the basis structure, and will be used in later proofs (this proof is very similar to the proof of Lenstra, Shmoys, and Tardos that gives a solution within a factor of two of optimal for the non–identical machine makespan problem (theorem 1 of [11])).

Theorem 1 *The number of split jobs in a basic solution to LGAP is at most the number of machines scheduled to capacity.*

PROOF: For each split job, we will associate exactly one machine used to capacity. The process associates each machine with at most one split job. Divide the nodes of the basis into cycle nodes and non–cycle nodes. Every node not on the cycle has a unique predecessor, giving the node one edge closer to the basis cycle for that component. Let j be a split job whose node is not on a cycle. Its corresponding machine node is any machine node with j as its predecessor. The existence of this node follows from the fact that a split job has degree at least two in the basis. Its uniqueness follows from the uniqueness of the predecessors. For j a split node on a cycle, let its corresponding node be the previous node on the cycle (where each cycle is arbitrarily oriented). This is a machine node by the bipartiteness of the basis graph. Uniqueness follows from the structure of cycles.

Finally, note that any machine associated with a job is used to capacity, by Lemma 1. $\square$

This shows that the linear relaxation schedules most of the jobs. If the unsplit jobs are now fixed, only a small number of jobs remain to be scheduled. We would like to schedule these jobs using the linear relaxation of the reduced problem. We avoid cycling by the following theorem.

Define a variable x_{ij} to be *useless* if $a_{ij} > b_i$. Clearly, deleting any useless variable does not affect GAP.

Theorem 2 *If there are any split jobs in a basic solution to LGAP, then fixing all variables of value 1 results in at least one variable becoming useless with respect to the new machine capacities.*

PROOF: Suppose there is a split job not on a cycle. Take one of maximal distance from the cycle in its component (measured in number of edges), say j. There is at least one edge with fractional value to a machine node farther from the cycle, say i. This machine node is otherwise only incident to unsplit jobs (by the choice of the split job). Therefore, after fixing unsplit jobs, the capacity of this machine is $x_{ij}a_{ij} < a_{ij}$ so x_{ij} is now useless.

Now suppose all split jobs are on the cycles. Consider any basis cycle. There are two cases:

(i) there is a machine node such that the sum of the two incident edges is less than 1. Let the machine node be i and the job nodes be j and k, where $a_{ij} \geq a_{kj}$. After fixing unsplit jobs the capacity of i is $a_{ij}x_{ij} + a_{ik}x_{ik} < a_{ij}$ so x_{ij} is useless.

(ii) all machine nodes have incident sum 1. If, for every machine node i and incident nodes j and k, $a_{ij} = a_{ik}$ then it is easy to see that the basis is dependent around the cycle. So there exists an i such that there are incident nodes j and k with $a_{ij} > a_{ik}$. The capacity of i is then $a_{ij}x_{ij} + a_{ik}x_{ik} = a_{ij}x_{ij} + a_{ik}(1 - x_{ij})$. Since one edge on the cycle has a non–integer value, all must, so $x_{ij} < 1$. The capacity of i is $< a_{ij}$ so x_{ij} is useless.

Note that if some machine has incident sum greater than 1, some machine on that cycle must have sum less than 1. □

This proof actually shows that the number of useless variables is at least the number of components of the basis that have a fractional edge.

This leads to the following heuristic for GAP, called LR–Heuristic for linear relaxation heuristic:

LR–Heuristic.
Input: an instance of GAP.
Output: a heuristic solution.

Step 0) Remove all useless variables; if no variables remain, then stop.

Step 1) Solve the linear relaxation, LGAP.

Step 2) Fix all variables that have value 1, deleting the job and updating the machine capacity. Go to step 0.

Theorem 3 *Step 1 of LR–Heuristic is executed at most $m + 1$ times.*

PROOF: By the previous theorem, at each execution of step 1, if there is still a split job, at least one edge becomes useless that was not useless before. But an edge can become useless only if at least one job is scheduled. By theorem 1, after the first iteration only m jobs remain to be scheduled, so the theorem follows. □

This heuristic can be effectively implemented using the data structures of Brown and McBride [2] or Nulty and Trick [15]. These implementations solve generalized networks orders of magnitude faster than general simplex codes. Assuming that n is much larger than m, only one large generalized network problem must be solved; all networks after the first have at most m jobs to be scheduled. Also, the value of the first generalized network problem solved is a valid lower bound on GAP; it solves the linear relaxation.

There are a number of variations on this approach. For instance, instead of fixing all variables with value 1, it is possible to delete all variables that would be useless if the variables were fixed but to not fix any of the variables. Since variables will be deleted each iteration (by theorem 2), this approach will not cycle. Such an method seems more flexible and may be better for some types of problems. In the computational results that follow, this heuristic acted essentially the same as LR–Heuristic so we do not continue with this approach.

Ideally, LR–Heuristic would not deviate from optimal by more than some constant factor. Unfortunately, it is easy to modify the NP–completeness result of [5] to show that no algorithm can guarantee closeness to optimality unless $P = NP$. This holds even for $m = 3$. However, an alternative measure of performance is number of jobs scheduled. We can bound the difference between the optimal number scheduled (for any cost function) and the number scheduled by LR–Heuristic. The obvious corollary to theorem 1 is a bound of m. The following improves that bound slightly in the case that there is a feasible solution to LGAP. Although the difference between m and $m-1$ seems minor, many applications have very small m, so the difference may be a large percentage.

Theorem 4 *If LGAP has a feasible solution then the difference between the number of jobs scheduled by LR–Heuristic and the optimal value is no more than $m-1$.*

PROOF: Examine the initial linear relaxation basis. If a slack arc associated with a machine is in the basis, then, by the count in theorem 1, the number of split jobs is at most $m-1$. If any job has degree three or more among the nonzero arcs in the basis, then some machine used to capacity is not matched in the proof of theorem 1, so the number of split jobs is at most $m-1$. In either case, the number of jobs that might not be scheduled is at most $m-1$.

Therefore, we can assume that every job has degree at most two and that there are no slack edges in the basis. Since LGAP has a feasible solution, every split job has degree exactly two. Since no slack edges are in the basis, there must be a basis cycle with at least two jobs. The claim is that there is an edge on this cycle that will not be marked useless. There must be a machine node i on this cycle with incident sum at least 1. Let the incident jobs be j and k with $a_{ij} < a_{ik}$. After fixing variables with value 1, the capacity of i is $a_{ij}x_{ij} + a_{ik}x_{ik} > a_{ij}$. So the edge (i,j) will not be marked useless. Therefore, the next generalized network to be solved will be able to schedule at least 1 job. Either it schedules a job or it schedules fractional jobs instead. But, as the proof of theorem 3 showed, if a fractional job is scheduled, at least one variable has value 1. The initial relaxation fails to schedule at most m jobs and the second relaxation schedules at least 1 job. □

3 Improving Heuristics

The solution of LR–Heuristic need not be optimal, of course; it may be possible to improve the solution. In this section we discuss methods for finding an improved solution.

A natural possibility is to use heuristics analogous to improving heuristics in the traveling salesman problem. The most well known improvement heuristic for the TSP is 2–opting, where two edges currently in the solution are exchanged for two other edges (still keeping a tour). If the resulting tour is better then it becomes the current solution. It is straightforward to generalize this to the GAP. Two jobs on different machines are swapped (if the machine capacities allow); if the resulting assignment is cheaper then it is adopted as the current solution. To this we can also add 1–opting: the assignment of a single job to a different machine. This has no analogous operation for the TSP but clearly is possible for the GAP.

Higher order k–opting tends to be very time consuming because the number of possible interchanges is of order n^k (although the number of interchanges performed is likely not polynomially bounded without some other restriction on the interchanges). One difference between the GAP and the TSP is that fixing a set of variables of the GAP still results in a GAP. We took advantage of this in LR–Heuristic and can take advantage of it also in an improving heuristic. Our idea is to fix a random part of the current solution and resolve using LR–Heuristic.

LR–Improve.
Input: an instance of GAP, solution x, fixed probability p.
Output: a new solution y.

Step 0) For each $x_{ij} = 1$, with probability p set $y_{ij} = 1$, update capacity of machine i, delete job j.

Step 1) Solve resulting problem with LR–Heuristic to get the remainder of solution y.

LR–Improve can be repeated, stopping with any rule (time limitations, gap between upper and lower bounds, etc.).

LR–Improve does not directly replace k–opting (for instance, if all but 2 jobs are fixed, LR–Improve might still not execute a feasible exchange). This is due to the use of a heuristic to schedule free jobs. As long as LR–Heuristic provides good solutions, LR–Improve has the possibility of moving out of non–optimal local minima.

There are many other ways of using the information found in the linear programming relaxation. Given a feasible solution (an upper bound), we can eliminate variables in LR–Improve by identifying variables that cannot be in any better solution. This routine replaces Step 1 in LR–Heuristic (here we assume the costs are integer).

LR–Elimination.
Input: an instance of GAP, partial solution x, upper bound u.
Output: modified GAP, with variables deleted that cannot be in a solution better than u, new solution y.

Step 0) Solve LGAP, with x fixed, to get solution y with cost $c(y)$ and dual solution π.

Step 1) For each edge (i, j), if $c(y) + c_{ij} - \pi_i - a_{ij}\pi_j > u - 1$ then delete (i, j).

The correctness of this deletion follows immediately from duality theory. It is useful to separate those variables eliminated by the initial linear relaxation (when no variables are fixed) and those eliminated later. The former variables can be ignored for the remainder of the heuristic; the latter only as long as x remains fixed. If enough variables get fixed permanently, then it is conceivable that optimality could be proved based solely on feasibility requirements.

4 Improving the Lower Bound

In the previous sections, we concentrated on trying to find good feasible solutions. In this section we outline how the basis structure of LGAP can be used to create better lower bounds.

One method of strengthening the lower bounds is to add constraints to LGAP that are *valid*. A constraint is valid if every feasible point of GAP satisfies it. Gottlieb and Rao ([7, 8]) have identified some classes of violated inequalities for the GAP and shown conditions under which these classes are facets. Unfortunately, they do not provide separation routines for their classes, limiting its computational usefulness. Here we use the linear relaxation to identify violated inequalities. In the next section, we will see how much the lower bound is improved by adding these constraints.

Note that the constraint set (1) is a set of knapsack constraints. Any constraint valid for an individual knapsack is clearly valid for GAP. The following theorem shows that there is a valid constraint for each useless edge found in theorem 2. Furthermore, this constraint is violated by the optimal solution to LGAP.

Theorem 5 *Let x_{ij} be an edge marked useless by theorem 2 and let S_i be the set of jobs assigned to machine i. Then the constraint*

$$\sum_{k \in S_i} x_{ik} + x_{ij} \leq |S_i| \tag{4}$$

is valid for GAP and the solution to LGAP violates this constraint.

PROOF: Validity follows directly from the argument to mark x_{ij} useless. Since $\sum_{k \in S_i} a_{ik} + a_{ij} > b_i$ it is not possible to schedule all of S_i and j on i. At most S_i can be scheduled.

Since $\sum_{k \in S_i} x_{ik} + x_{ij} = S_i + x_{ij}$ and $x_{ij} > 0$, the solution violates this constraint. $\square$

These constraints are similar to, but not identical to, the knapsack constraints found by Gottlieb and Rao in their examination of valid constraints of the GAP from LGAP.

We can strengthen the constraints found by removing variables from S_i until $S_i \cup \{j\}$ is a *minimal cover* (a minimal set that cannot be scheduled on the machine) for constraint i (see the text by Nemhauser and Wolsey [14] for a survey of strengthening valid inequalities for the knapsack problem).

There are more constraints that can be found directly from the linear relaxation. The following class of constraints is violated by the linear relaxation and is not prohibited by the constraints (4).

Theorem 6 *Let i be a machine node not on the cycle such that the edge (i, j) to its predecessor is fractional. Let T be the edges of the subtree below i in the basis (i.e. farther from the cycle) and let J be the jobs below i. Then the constraint*

$$\sum_{(k,l)\in T} x_{kl} + x_{ij} \leq |J| \tag{5}$$

is valid for GAP and is violated by the linear relaxation.

PROOF:

Violation: Every job in J is completely scheduled using the edges of T so $\sum_{(k,l)\in T} x_{kl} = |J|$. Since $x_{ij} > 0$, the constraint is violated.

Validity: There are $|J| + 1$ jobs associated with the constraint, so the value of the left hand side is at most $|J| + 1$. If it equaled $|J| + 1$ then every job could be scheduled using just $T \cup (i, j)$. But T schedules $|J|$ jobs leaving capacity at i of $a_{ij}x_{ij}$. Any other scheduling of those jobs in T leaves no more than $a_{ij}x_{ij} < a_{ij}$. Since j can only be scheduled on i no more than $|J|$ jobs can be scheduled. $\square$

These constraints are a generalization of the first set: taking the machine node so that T contains no fractional edges gives the constraints of Theorem 5. The following shows that these constraints are not implied by LGAP together with (4).

Theorem 7 *There exist instances where the constraints (5) are more restrictive than LGAP together with the constraints (4)*

PROOF:

Consider the instance in Figure 1(a), where the numbers are the node numbers and all edges not shown are useless. The filled circles are job nodes; the open ones machines. For every nonuseless assignment, let the size of the

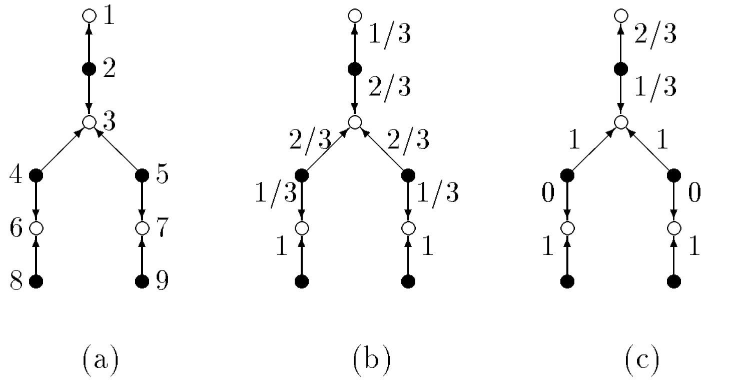

Figure 1: Violated extended knapsack

job on the machine be 3, except job 2 on machine 3 has size 6. The capacities of machines 6 and 7 are 4; machine 3 is 8; and machine 1 is 50. It is easy to assign costs so that Figure 1(b) gives the linear relaxation, where the numbers are the flows. Using node 3, we deduce that the sum of the edges of the tree below node 2 is at most 4. Constraints of type (4) are generated at nodes 6 and 7. Figure 1(c) gives a solution with sum on the tree of 4 1/3 which satisfies the knapsack constraints on nodes 6 and 7, as well as the constraints of LGAP, proving the theorem. □

An alternative way of generating these constraints is to identify machine nodes not on a cycle and incident to a fractional edge. If S_i is the set of nodes incident to i via a nonzero edge in the basis then $\sum_{\{j \in S_i\}} x_{ij} \leq |S| - 1$ is a valid inequality. Linear combinations of these inequalities make up equation (5). These inequalities need not be violated by the linear relaxation, however. The previous theorem is formulated to show that they are "tight enough." That is, they are not so loose that adding them to the inequalities (4) gives nonviolated constraints.

Gottlieb and Rao provide another set of constraints which are valid inequalities for the GAP. Some instances of this set, which they call the cycle inequalities, can be identified directly from the final basis as edges are marked useless. Given a basis, a valid cycle inequality can be generated when the following occur:

(a) there exists at least one machine node on a basis cycle both of whose cycle edges are marked useless, and

(b) every edge on the cycle has positive flow.

In this case, let $p = \lfloor (M-1)/2 \rfloor$, where M is the set of machine nodes of type (a) on a cycle, and let E be the set of edges on the basis cycle together with every edge with value 1 incident to a machine node on the cycle. Let J be the set of jobs incident to an edge of E. The constraint

$$\sum_{\{(i,j)\in E\}} x_{ij} \leq J - p - 1 \tag{6}$$

is a valid inequality for GAP. It is straightforward to prove validity by showing that this is a special case of the cycle inequalities of [7]. The linear programming relaxation does not necessarily violate this constraint, but it will if the flow on the cycle is sufficiently high (for instance, if every job on the cycle is scheduled just by the machines on the cycle). An interesting open question is to find all such inequalities violated by the linear relaxation. For a more general definition of cycle inequalities, see [7].

These constraints have the nice property that some instances of them are not derivable from just LGAP and all the facets of the individual knapsack constraints. Gottlieb and Rao provide an example of this fact. The following example shows that this is true even for violations found from the linear relaxation.

Theorem 8 *There exist instances where the constraints (6) generated from the linear relaxation are not implied by LGAP together with every facet of the individual knapsack constraints.*

PROOF:
Consider the example in figure 2, where the labels on the edges in 2(a) are the sizes of the jobs on the machines and the labels on the nodes (in boxes) are the capacities of the machines. Costs can be assigned so that 2(b) gives the linear relaxation. This generates a constraint for the edges of 2(b), with a maximum of 5 on those edges. Finally, if the flow of every edge of 2(c) is 1/2, then the solution satisfies LGAP as well as every knapsack facet and has value 5.5 on E. □

Adding these constraints to LGAP will normally provide a better relaxation to GAP. The resulting linear program can be solved using either

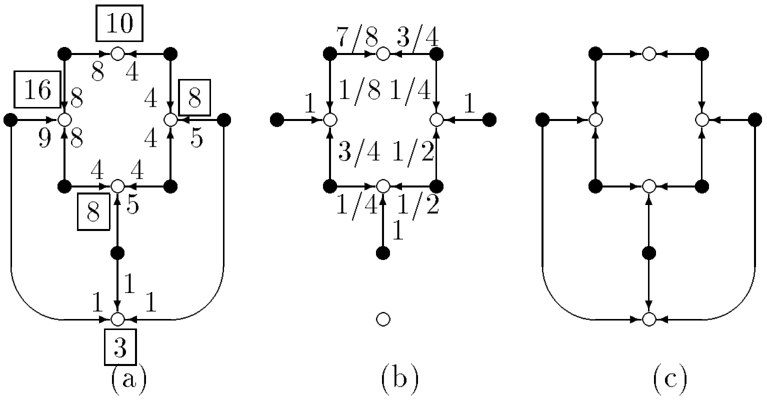

Figure 2: Violated cycle inequality

a general linear programming code or a generalized network with side constraints code (McBride [13]), or the constraints can be relaxed by Lagrangian relaxation (for example, Fisher [3]). Because of the possibility of generating constraints not derivable from the individual knapsack constraints, it is possible that the resulting lower bound will be better than that of Fisher, Jaikumar and Van Wassenhove [5], who give bounds equivalent to adding all the constraints deducible from the individual knapsacks. Another advantage of this approach is that no knapsack problem need be solved. If the capacities of the machines are large, these knapsacks can be very time consuming. The violated inequalities, on the other hand, are a simple byproduct of the linear programming relaxation. Combined with the variable elimination method presented earlier, this approach has the possibility of completely solving some instances without resorting to branch and bound. Note, however, that the method of Fisher et. al. is equivalent to generating all knapsack constraints, while the method here generates only a few.

5 Computational Results

We have examined the linear programming relaxation of the GAP from a theoretical point of view. This has led to heuristics that optimally schedule a large number of jobs, and schedule the rest as well as possible. We also have an improvement heuristic, a method for eliminating variables, and an improved lower bound. In this section we examine how well these ideas work in practice.

We compare our heuristics with two others suggested in the literature. The first is the greedy algorithm, where each job is sequentially assigned to its "best" machine. In this test, modifying a suggestion of Martello and Toth [12], we define the best machine for a job to be the one that has minimum cost divided by the remaining capacity of the machine. In a later test, we will modify the definition of "best machine" for highly congested systems. The goal is to balance low cost with machine usage. Since the jobs are assigned sequentially, we can update the machine capacities to reflect assigned jobs. Our second heuristic, called savings–regret, finds the difference between the best and second best machine for each job. The job with the largest difference is assigned to its best machine (this is essentially the heuristic in [12]). The results for all heuristics are then improved by 1– and 2–opting. Other heuristics are discussed by Jacobs ([9]) and Klastorin ([10]). Furthermore, it would be interesting to compare this heuristics with the optimization based routines of [5] and [12]. These methods often find very good or optimal solutions early in the branch and bound search tree.

To generate our problems, we begin with the same distributions used in previous studies [12, 5]. These distributions are denoted A, B, C, and D. The results in those papers concentrated on exact solutions, so the problem sizes examined are naturally quite small. This testing concentrates on heuristic solutions, so our sizes are much larger. Unfortunately, it turns out that these generators create easy problems as the sizes get larger. The difficulty is that the machine capacities are too large. In order to make the results more relevant, we modify the capacities to make the problems more challenging. The first distribution, corresponding to distributions A and B, generates a_{ij} and c_{ij} uniformly between 5 and 25 and 10 and 50 respectively. Each machine has the same capacity. Previous work has used a combination of the size required to make the problem trivial (by allowing each job to use its cheapest machine) and a target size on the machine. This target size is a constraint on the average size of an assignment. Sufficient machine capacity is assigned to force the average size of an assignment to be less than the target size. Given a target size k, this value is kn/m per machine. As problems get larger, the size that makes the problem trivial gets quite large. Therefore, we use only a target size. We tested at three levels: highly constrained, moderately constrained, and weakly constrained. This corresponds to k of 7.5, 10, and 15 respectively. This generator is denoted generator I.

Our second generator, generator II, is identical to generator C in the literature. Sizes and costs are generated as in generator I, but the capacity of machine i is set to be some fraction of $\sum_j a_{ij}/m$. Again we have three levels of constraint: highly constrained, with the fraction set to .4, moderately constrained with a fraction of .6, and weakly constrained with a fraction of .8 (corresponding to generator C).

Our final generator, generator III, is identical to generator II, except the sizes are generated uniformly between 1 and 100 and the cost is set to 111 minus the size plus a random variable generated between -10 and 10. The machine sizes are the same, as are the three levels of constraint. The weakly constrained problems are exactly those of generator D.

For each generator, problems were generated with five different sizes: 20 jobs and 5 machines, 50 jobs and 10 machines, 100 jobs and 10 machines, 100 jobs and 30 machines, and 500 jobs and 100 machines.

Tables 1 through 3 present the results. The numbers reported in these tables give the solution quality in terms of percentage over the linear programming lower bound. Twenty five instances were run for each problem size; the numbers reported are the average of the results. Unscheduled jobs were charged twice the largest cost in the network (i.e. charged 100 in I and II and 242 in III). For LR–Improve, the initial solution was that of LR–Heuristic and five iterations were made; p was set to 0.3.

LR–Heuristic consistently provides better solutions, both in terms of percentage over lower bound and in number of jobs scheduled. For some types of problems, notably the highly constrained problems, LR–Heuristic's improvement was very large. The improvement was consistent across problem generators and instances: LR–Heuristic found a solution at least as good as the better of the greedy and savings regret solution in 1057 of the 1125 instances represented in the tables. LR–Improvement provided a modest, but steady improvement over LR–Heuristic.

It should be noted that these results are slightly misleading due to our handling of solutions that did not schedule all of the jobs. Table 4 lists all sizes where at least one heuristic did not schedule all the jobs for at least one instance. The table gives the average number not scheduled for each heuristic. For these heuristics and classes of problems, the percentage above the lower bound can be made arbitrarily high simply by increasing the penalty for not assigning a job. It is clear that LR–Heuristic is less susceptible to this difficulty, both computationally and through theorem 4.

To further explore the effect of varying the effect of the capacity con-

Size (job,mach.)	Level of Constraint	Percentage over lower bound			
		Greedy	SavReg	LRHeur	Improve
20,5	High	32.39	25.15	15.38	14.97
	Med.	22.98	21.26	13.17	11.13
	Low	4.75	4.56	3.04	2.11
50,10	High	39.78	36.30	14.55	11.19
	Med.	13.73	8.83	8.20	6.59
	Low	3.61	3.60	1.91	1.59
100,10	High	45.35	35.52	5.85	5.07
	Med.	7.06	6.70	3.55	3.10
	Low	1.61	1.72	0.60	0.49
100,30	High	24.66	16.83	14.24	13.19
	Med.	10.42	9.99	3.20	2.15
	Low	3.92	3.96	3.70	2.20
500,100	High	7.65	6.73	5.37	5.31
	Med.	4.73	4.67	2.19	2.16
	Low	0.79	0.80	0.27	0.22

Table 1: Solution Quality (Generator I)

Size (job,mach.)	Level of Constraint	Percentage over lower bound			
		Greedy	SavReg	LRHeur	Improve
20,5	High	24.69	22.60	10.30	9.83
	Med.	33.87	29.20	16.54	14.77
	Low	16.01	13.29	10.46	8.83
50,10	High	42.22	38.06	10.26	8.93
	Med.	24.04	17.97	11.42	8.85
	Low	9.71	8.34	6.18	5.34
100,10	High	42.33	39.16	4.14	3.66
	Med.	13.09	7.32	4.18	3.60
	Low	5.93	5.74	2.62	2.29
100,30	High	70.32	64.57	28.03	26.54
	Med.	14.46	11.29	12.20	10.15
	Low	8.26	8.04	5.57	4.77
500,100	High	69.59	51.73	26.24	25.00
	Med.	6.34	6.06	3.26	3.26
	Low	3.00	3.02	1.64	1.61

Table 2: Solution Quality (Generator II)

Size (job,mach.)	Level of Constraint	Percentage over lower bound			
		Greedy	SavReg	LRHeur	Improve
20,5	High	26.37	17.36	11.56	9.17
	Med.	20.06	15.47	7.11	5.57
	Low	8.99	8.31	5.82	4.78
50,10	High	15.50	14.57	3.15	2.47
	Med.	7.60	7.41	3.29	2.92
	Low	8.57	8.78	3.93	3.60
100,10	High	11.26	10.25	1.29	1.10
	Med.	6.66	6.73	1.56	1.41
	Low	7.59	7.74	1.99	1.69
100,30	High	6.86	7.01	3.18	2.92
	Med.	8.35	8.44	4.01	3.80
	Low	9.24	9.12	4.78	4.51
500,100	High	7.65	7.77	3.10	3.10
	Med.	6.66	6.86	3.01	3.01
	Low	8.18	8.14	3.77	3.77

Table 3: Solution Quality (Generator III)

Generator	Size (job,mach.)	Level of Constraint	Greedy	Savings Regret	LRHeur	Improve
I	20,5	High	4.4	3.7	2.6	2.5
		Med.	1.1	1.0	0.3	0.2
	50,10	High	6.2	5.8	1.4	0.8
	100,10	High	14.9	11.0	0.9	0.6
	100,30	High	0.4	0.2	0.0	0.0
II	20,5	High	7.9	7.6	5.6	5.5
		Med.	3.2	2.6	1.3	1.2
		Low	0.2	0.1	0.0	0.0
	50,10	High	17.0	16.1	7.8	7.4
		Med.	1.5	0.9	0.3	0.0
	100,10	High	33.4	32.1	11.7	11.6
		Med.	1.4	0.4	0.0	0.0
	100,30	High	23.6	22.3	9.7	9.2
	500,100	High	51.2	38.5	16.8	14.6
III	20,5	High	3.3	2.1	1.3	1.0
		Med.	1.4	1.0	0.2	0.1
	50,10	High	3.0	2.7	0.1	0.0
	100,10	High	3.7	3.0	0.0	0.0

Table 4: Average Number Not Scheduled

Figure 3: Effect of target size on solution quality.

straints, the 100 job, 30 machine problem with generator I was rerun with many more target sizes. The results are in figure 3. LR–Heuristic is much less sensitive to the degree of capacity constraint, while the other two heuristics degrade rapidly for highly constrained problems.

As for computation time, as might be expected, the linear programming based heuristic requires more time, although the time required for 2–opting means that the difference is not overwhelming. The average times for the heuristics are given in table 5 (averaged over the data sets on a SUN SPARC–1 workstation).

Next, we examine the effect of adding the valid inequalities generated from the initial basis. These constraints must cut off the relaxation solution, but it is not known how much they will improve the lower bound. For the instances in the smaller two data sets, the cuts generated by constraint sets (4) and (5) were added to the linear relaxation and the resulting linear

Jobs	Mach-ines	Greedy	Savings Regret	LR–Heuristic	Improve
20	5	0.0	0.0	0.1	0.2
50	10	0.1	0.3	0.5	1.5
100	10	0.6	0.8	1.4	4.8
100	30	0.8	1.2	3.4	7.8
500	100	16.8	26.3	81.6	202.0

Table 5: Computation time (seconds)

Size (job,mach.)	Level of Constraint	L.P. with Cuts	Integer Program	LRHeur
20,5	High	2.3	10.0	14.8
	Med.	1.2	6.8	11.5
	Low	1.2	2.1	2.8
50,10	High	1.0	—	13.4
	Med.	0.8	—	8.9
	Low	0.3	—	1.7
100,10	High	0.3	—	5.5
	Med.	0.2	—	3.6
	Low	0.1	—	0.6
100,30	High	1.1	—	16.0
	Med.	0.7	—	7.9
	Low	0.3	—	2.4

Table 6: Comparison with Cuts and Optimal Integer Solution (Generator I)

program resolved. In a production code, a network with side constraints program would be used, like that described [13]. Here we simply used a general linear programming code. The resulting improvement in the lower bound is reported in table 6, in the column labeled "L.P. with Cuts". Again, the entries are in percent over lower bound, and the values are the average of 10 runs.

The improvement is noticeable, but not as large as might be hoped. An improved cut generating method for GAP will have to either find more constraints from the linear relaxation or use constraints not available from the initial basis. This important area of research has been begun by Gottlieb and Rao ([7, 8]). Further research on valid inequalities and, especially, separation algorithms is needed.

Finally, we address the question of how good the heuristic is in absolute terms. For each of the smallest problems, we resolved each instance using the integer programming capabilities of LINDO [18]. The resulting integer solutions (again in terms of percentage over lower bound) are presented in Table 6 in the column "Integer Program". It is clear that LR–Heuristic finds reasonably good answers, but there is room to improve the lower bound generated.

In conclusion, repeatedly solving the linear relaxation not only has some nice theoretical properties but seems very competitive in practice. It provides very good solutions along with an upper bound on the deviation from optimality. With network data structures, large problems can be solved very quickly, enhancing the usefulness of this method.

References

[1] J.F. Benders and J.A.E.E. van Nunen, "A property of assignment type mixed integer linear programming problems," *O.R. Letters*, 2, 47–52 (1982).

[2] G.G. Brown and R. McBride, "Solving generalized networks," *Management Science*, 20, 1497–1523 (1985).

[3] M.L. Fisher, "An applications oriented guide to Lagrangian relaxation," *Interfaces*, 15, 10–21 (1985).

[4] M.L. Fisher and R. Jaikumar, "A generalized assignment heuristic for vehicle routing," *Networks*, 11, 109–124 (1981).

[5] M.L. Fisher, R. Jaikumar, and L. Van Wassenhove, "A multiplier adjustment method for the generalized assignment problem," *Management Science*, 32, 1095–1103 (1986).

[6] B.L. Golden and W.R. Stewart, "Empirical analysis of Heuristics," 207–249 of *The Traveling Salesman Problem*, edited by E.L. Lawler, J.K. Lenstra, A.H.G. Rinooy Kan, D.B. Shmoys, John Wiley and Sons, New York (1985).

[7] E.S. Gottlieb and M.R. Rao, "The generalized assignment problem: Valid inequalities and facets," *Mathematical Programming*, 46, 31–52 (1990).

[8] E.S. Gottlieb and M.R. Rao, "$(1, k)$ Configuration facets for the generalized assignment problem," *Mathematical Programming*, 46, 53–60 (1990).

[9] C.D. Jacobs, *The Vehicle Routing Problem with Backhauls*, Ph. D. dissertation, Georgia Institute of Technology (1987).

[10] T.D. Klastorin, "An effective subgradient algorithm for the generalized assignment problem," *Computers and Operations Research*, 6, 155–164 (1979).

[11] J.K. Lenstra, D.B. Shmoys, and E. Tardos, "Approximation algorithms for scheduling unrelated parallel machines," *Mathematical Programming*, 46, 53–60 (1990).

[12] S. Martello and P. Toth, "An algorithm for the generalized assignment problem," *Operational Research '81*, J.P. Brams (editor), North–Holland, New York (1981).

[13] R.D. McBride, "Solving embedded generalized network problems," *European Journal of Operational Research*, 21, 82–92 (1985).

[14] G.L. Nemhauser and L.A. Wolsey, *Combinatorial and Integer Programming*, John Wiley, New York (1988).

[15] W.G. Nulty and M.A. Trick, "GNO/PC generalized network optimization system," *O.R. Letters*, 2, 101–102 (1988).

[16] G.T. Ross and R.M. Soland, "A branch and bound algorithm for the generalized assignment problem," *Mathematical Programming*, 8, 91–103 (1975).

[17] G.T. Ross and R.M. Soland, "Modeling facility location problems as generalized assignment problems," *Management Science*, 24, 345–357 (1977).

[18] L. Schrage, *Linear, integer, and quadratic programming with LINDO*, The Scientific Press, Redwood City, CA (1986).