SCHEDULING MULTIPLE VARIABLE–SPEED MACHINES

MICHAEL A. TRICK

CARNEGIE MELLON UNIVERSITY, PITTSBURGH, PA

April, 1990; Revised October, 1990; November, 1991; June, 1992

Abstract. We examine scheduling problems where we control not only the assignment of jobs to machines, but also the time used by the job on the machine. For instance, many tooling machines allow control of the speed at which a job is run. Increasing the speed incurs costs due to machine wear but also increases throughput. We discuss some fundamental scheduling problems in this environment and give algorithms for some interesting cases. Some cases are inherently difficult so for these we give heuristics. Our approach illustrates the exploitation of underlying network structure in combinatorial optimization problems. We create heuristics that optimally schedule a large portion of the jobs and then attempt to fit in the remainder. This also gives a method for quickly finding valid inequalities violated by the linear relaxation solution. For the problem of minimizing the sum of makespan and production costs, a rounding heuristic is within a constant factor of optimal. Our heuristics are compared to other, classical, heuristics.

Traditional sequencing and scheduling models assume that the time a job requires on a machine is not under control: either the time is fixed, or it is a random variable determined by outside forces. Some problems require not only the assignment of job

A previous version of this paper was presented at the Conference on Integer Programming and Combinatorial Optimization, Waterloo, 1990, and was contained in the proceedings.

to machine but also the choice of processing time, reflecting physical capabilities of the machines or extended possibilities in the model. In this paper, we examine some optimally solved special cases and some heuristics in this environment, which we call *variable–speed scheduling.*

One example of variable–speed machines occurs in the scheduling of tooling machines. Tooling machines take pieces of wood, plastic, or metal and, through cutting and planing, make smaller pieces of the desired shapes and sizes. In such applications, tool wear is considerable. It is possible to decrease tool wear by running the machines at lower speeds (shallower cuts, more planing passes) but this increases the time spent by the piece on the machine.

This topic has begun to receive more attention due to its applicability to flexible machine scheduling with variable–speed machines. Many articles have been written developing methods for optimally cutting one piece on one machine. For a survey, see Gray, Seidmann and Stecke (1988), with other papers being Phillipson and Ravindran (1979) and Malakooti and Deviprasad (1989). Schweitzer and Seidmann (1989)(1991) have examined models with many jobs and machines and have investigated nonlinear cost functions and queuing effects. In another line of work, Daniels and Sarin (1989) and Vickson (1980a)(1980b) develop algorithms for precedence constrained single machine scheduling with a variety of optimality conditions with cost functions that are linear in the processing time. In this paper, we also concentrate on linear cost functions (though all the results are applicable to piecewise–linear convex cost functions). We concentrate on the fundamental problem of multiple machine scheduling without precedence constraints.

In addition to solving an interesting and useful problem, a goal of this paper is to illustrate the exploitation of embedded network structure in combinatorial optimization. We choose a formulation of this problem that has a generalized network problem as a linear relaxation. We can then find optimal solutions for the relaxation very quickly. More importantly, the structure of the basis of the relaxations contains an enormous amount of information. We can optimally schedule a large portion of the problem, leaving just a few jobs to be rescheduled. Furthermore, we can use the relaxation to heuristically reschedule the remainder. We also find violated inequalities directly from the basis structure.

We look at two types of models. In the first, the *capacitated machine* model, the capacity of each machine is fixed in advance. We have n jobs (indexed by j) and m machines (indexed by i). The machines are not assumed to be identical, or otherwise similar in characteristics. Each job must be assigned to the machines. If job j is assigned to machine i, the maximum amount of time the machine will take is u_{ij} and a nonnegative cost of c_{ij} is incurred. The job can take as little as l_{ij} units of time on the machine. For each unit less than the maximum, a nonnegative cost of s_{ij} must be paid. Machine i has b_i units of time available. Each job is assigned to exactly one machine, so the jobs are *indivisible.* The objective is to minimize the total cost.

In the second model, the *makespan model*, the amount of time available is not known in advance. Instead, we are concerned with the tradeoff between makespan (the maximum amount of time used by any machine) and operating costs. We assume that the operating costs have been scaled so that the objective is to minimize the sum of makespan and operating costs. The remainder of the problem is the same as above. Potts (1985) and Lenstra, Shmoys and Tardos (1990) have examined this problem without variable speeds and operating costs. Potts (1985) gave a heuristic based on fixing the integral part of a linear relaxation solution and enumerating on the remainder. Lenstra, Shmoys and Tardos (1990) were then able to show a simple rounding rule that gives a solution that is within a factor of two of optimum. We extend their analysis to this more general case and give a polynomial time heuristic that is within a factor of 2.6181 of optimum.

We begin by giving algorithms for finding the optimal solution for one very simple case: one capacitated machine. Although the algorithm is straightforward (it is simply the greedy algorithm), the solution structure is used as a building block for more difficult cases.

We then examine the multiple machine, indivisible job case for capacitated machines. The case where there is no flexibility in the machine speeds (i.e. $l_{ij} = u_{ij}$ for all i and j) is the generalized assignment problem. Because this problem is NP–hard, the variable–speed problem is as well. In Trick (1991), we examine the linear relaxation of the generalized assignment problem and find much structure. This leads to a heuristic with good behavior, both theoretically and practically. In this case as well, the linear relaxation provides a tremendous amount of information. We examine this structure in some detail and provide a good heuristic for the variable speed scheduling problem. We continue by finding violated valid inequalities for this problem that can be used to find improved lower bounds.

We use some of these results in our examination of the makespan model. Here we present a heuristic that is within a constant factor of optimum.

We conclude with computational results, comparing our heuristics with standard greedy and savings–regret heuristics for this problem. Our heuristics are robust, accurate, and fast.

1. Single Capacitated Machine

We begin with the single machine case. A formulation of this problem is (in this section we suppress the machine subscript i):

Minimize $\sum_j (c_j + s_j(u_j - x_j))$

Subject to

$\sum_j x_j \leq b$

$l_j \leq x_j \leq u_j$ for all j.

The x_j are interpreted as the amount of time spent by job j on the machine.

This problem is simply a linear knapsack problem with lower and upper bounds. There is a very simple solution procedure. First, determine if the instance is feasible by checking that the sum of the minimum times on the machine is no more than the time available on the machine. If that is the case, we can assign every x_j its minimum value. Our objective is to assign the remaining time of the machine so as to maximize our profit. The solution procedure is well known:

ALGORITHM Single Machine

Input: Single machine problem with n, c_j, s_j, l_j, u_j, and b.

Output: Optimal x_j.

```
Step 0) Sort the items so that $s_1 \geq s_2 \geq s_3 \geq \ldots \geq s_n$
Step 1) Let $excess = b - \sum_j l_j$, $k = 1$. If $excess < 0$, stop, problem is infeasible.
Step 2) If $u_k - l_k > excess$, go to step 4. Else go to step 3.
Step 3) Let $x_k = u_k$, $excess = excess - (u_k - l_k)$, $k = k + 1$. Go to step 2.
Step 4) $x_k = l_k + excess$, $x_{k'} = l_{k'}$ for all $k' > k$. Stop.
```

In this algorithm, all variables are set to their lower bound and then increased in order of their variable costs. In the solution, every job except one is processed using either its minimum time or its maximum time, and those that use their minimum times are those with the smallest variable cost per unit time.

Lemma 1 follows directly from this algorithm.

Lemma 1. *For any optimal solution to the single capacitated machine problem, the variables can be partitioned into three sets (any of which may be empty): A set U of variables at their upper bound, a set L with values at their lower bound, and a single variable j^* with a value in between. Furthermore, $s_j \geq s_{j^*} \geq s_k$ for any $j \in U$ and $k \in L$.*

2. Multiple Capacitated Machines

We now move to the multiple machine problem with indivisible jobs. One formulation of this problem is:

(IP1)

Minimize $\sum_i \sum_j c_{ij} z_{ij} + \sum_i \sum_j s_{ij}(u_{ij} z_{ij} - x_{ij})$

Subject to

$\sum_j x_{ij} \leq b_i$ for all i,

$\sum_i z_{ij} = 1$ for all j

$l_{ij} z_{ij} \leq x_{ij} \leq u_{ij} z_{ij}$ for all i, j,

$x_{ij} \geq 0$ for all i, j,

$z_{ij} \geq 0$ and integer for all i, j.

The z_{ij} are interpreted as the fraction of job j assigned to machine i and x_{ij} is time job j spends on machine i. We assume that IP1 represents a feasible problem.

Practically, this can be guaranteed by adding an extra artificial machine with large capacity and suitably large costs.

While the above formulation is adequate for using standard branch and bound techniques, there is an alternative formulation that is more useful for our purposes. In this formulation, we write the problem as a *integer generalized network*. In doing so, we are able to take advantage of the underlying network structure to create good heuristics.

Consider the following formulation (with variables y_{ij} and y'_{ij}):

(IP2)

Minimize $\sum_i \sum_j c_{ij} y'_{ij} + \sum_i \sum_j (c_{ij} + s_{ij}(u_{ij} - l_{ij})) y_{ij}$

Subject to

$\sum_j (l_{ij} y_{ij} + u_{ij} y'_{ij}) \leq b_i$ for all i,

$\sum_i (y_{ij} + y'_{ij}) = 1$ for all j,

$y_{ij} + y'_{ij} \in \{0, 1\}$ for all i, j,

$y_{ij}, y'_{ij} \geq 0$ for all i, j.

(Note that the constraints $y_{ij} + y'_{ij} \in \{0, 1\}$ are not quite in standard form. We could standardize by adding a new variable, but that will not be necessary.) The y_{ij} are interpreted to be the fraction of job j using the minimum time on machine i and y'_{ij} the fraction using the maximum time. Since a job using an intermediate time can be thought of as some portion using the maximum time and the rest the minimum time, the equivalence of IP1 and IP2 is obvious.

The variable–speed scheduling problem as we have presented it is formally an NP–hard problem, as the following lemma shows:

Multiple Machine Variable–Speed Scheduling

GIVEN: Problem data: $m, n, c_{ij}, s_{ij}, l_{ij}, u_{ij}, b_i$ and a goal value K.

QUESTION: Is there an assignment of jobs to machines and processing times to jobs such that

(1) Each job is assigned to exactly one machine
(2) Each job has a processing time between its lower and upper values for that machine,
(3) The total time on each machine is no more than b_i, its capacity, and
(4) The total cost (fixed cost plus variable cost) is no more than K?

Lemma 2. *Multiple Machine Variable–Speed Scheduling is NP–complete.*

Proof. Given a candidate solution, it is easy to determine whether it satisfies the above conditions, so the problem is in NP.

If $l_{ij} = u_{ij}$ for all i and j, then the problem is the Generalized Assignment Problem, which is NP–Complete by the reduction in Fisher, Jaikumar and Van Wassenhove (1986), so our problem is also. $\square$

By examining the reduction (from partition), it is also possible to show the following corollaries.

Corollary 1. *It is NP–complete to determine if an instance of the Multiple Machine Variable–Speed Scheduling problem has a feasible solution.*

Corollary 2. *For any $k \geq 1$, unless $P = NP$, there is no polynomial time heuristic for the Multiple Machine Variable–Speed Scheduling problem that is guaranteed to be within a factor of k of optimal.*

We are interested in the linear relaxations of IP1 and IP2. In IP1, the linear relaxation is formed by removing the integrality restrictions on z_{ij}. We call the resulting linear program LP1. Similarly, in IP2 we remove the integer restrictions on $y_{ij} + y'_{ij}$ to form LP2. In the next section, we examine the structure of the optimal solutions to LP2.

2.1. Linear Relaxation Structure. By working with LP2 rather than LP1, we can get more information out of the structure of the optimal solution. Because each variable appears in just two constraints (plus non–negativity), LP2 is a generalized network problem (or network with gains and losses). The form of this generalized network is illustrated in Figure 1. It is a bipartite graph with one side of the partition representing jobs and the other representing machines. It is a multigraph with precisely two edges from each job–node to each machine–node.

—Figure 1 around here—

FIGURE 1. Generalized network for variable–speed problem

If we solve this generalized network problem with a simplex algorithm we will find an optimal basis. A basis can be seen as a subgraph of the original generalized network. This subgraph also has a particular structure. In particular, the arcs that form the basis form a set of one–trees (trees with one extra edge), provided we treat slack variables as degenerate cycles. From restrictions on the number and form of basis cycles, it is easy to prove the following about the structure of any basic feasible solution:

Theorem 1. *The subgraph representing any basic solution to LP2 is a set of connected components, where each component has exactly one of:*

(1) *a machine not used to capacity;*

(2) *a job assigned to a machine using a time between its maximum and minimum; or*

(3) *a cycle of length at least four nodes of alternating job–nodes and machine–nodes.*

Proof. Each of these form a basis cycle. In case 1, the cycle corresponds to the degenerate cycle. In case 2, the cycle consists of a single job node and a single machine node and both arcs between them. In case 3, the cycle consists of four or more nodes, which must alternate due to the bipartite structure of the graph. These are the only ways to form a basis cycle. As shown in Brown and McBride (1985), there is exactly one basis cycle per component. □

We call a job using a time between its maximum and minimum time an *intermediate job*. We call a job that is partially assigned to more than one machine a *split job*. Such a job is incident to at least two machine–nodes in the basis, though not every job incident to more than one machine–node is a split job. The latter fact is due to the possibility of a degenerate basis, where there are basic arcs with zero flow. It is possible to rephrase Theorem 1 in terms of the graph of arcs with non–zero flow rather than the graph of basic arcs by including components that are trees.

Based on this theorem, there are a number of corollaries regarding basic solutions to LP2.

Corollary 3. *For every basic feasible solution to LP2, the following hold:*

(1) *There is at most one intermediate job on any machine.*

(2) *Every job uses an intermediate time on at most one machine.*

(3) *If a machine has an intermediate job assigned to it, then the machine is used to capacity.*

Proof. If any of these did not hold then the conditions of Theorem 1 would be violated. □

Also, with methods similar to those used in Lenstra, Shmoys and Tardos (1990) and Trick (1991), we can bound the number of split and intermediate jobs.

Corollary 4. *The total number of split or intermediate jobs is no more than the number of machines scheduled to capacity.*

Proof. We prove this corollary by associating each split or intermediate job with a machine scheduled to capacity. Each machine scheduled to capacity is associated with at most one job, so the corollary follows.

Take a split job not on a basis cycle. Since it has degree at least two in the basis, associate with it any machine node one further from the basis cycle. For any split job on a basis cycle, orient the cycle arbitrarily (but consistently for every node on the cycle). Associate with each split job the next machine node on the cycle. Finally, for any intermediate node, associate it with the machine on which it uses an intermediate size. Every split and intermediate job is now associated with a machine; no machine is associated with more than one job; no machine not used to capacity is associated with any job. □

2.2. A linear relaxation based heuristic. We now address the problem of finding a good heuristic for the integral restricted problem. Of course, by Corollaries 1 and 2, such a heuristic can neither guarantee a fixed deviation from optimality nor even be certain of returning a feasible solution (even if the instance is feasible). Despite this, our approach has a number of nice properties, including a limit on the number of unscheduled jobs and good empirical behavior.

Our approach is to use the results from the previous section as a starting point. We know by Corollary 4 that if there are a large number of jobs and a small number of machines then almost all jobs are assigned to one machine each. By Corollary 4, at most m jobs are split jobs. This leaves us just a small number of jobs to schedule. We therefore examine a heuristic that takes a fractional solution, fixes all assignments for jobs uniquely assigned to a machine and creates a new problem called the *remaining problem*. The remaining problem has only those jobs not uniquely assigned by the fractional solution. The capacity for each machine is equal to the amount not used by the fixed jobs.

To solve the remaining problem, it would be natural to use linear programming again. It is important to ensure that we do not cycle, generating the same solution over and over. We can do this by modifying the instance in a way that keeps the integer program the same, but changes the linear relaxation.

Consider a job j and machine i. If the upper bound $u_{ij} > b_i$, then clearly we can reduce u_{ij} to b_i without changing the integer program. Furthermore, if the resulting upper bound is less than the corresponding lower bound, then the variable can be deleted altogether. This process does change the linear relaxation, however. We call this process *variable reduction*, or *reduction* for short.

Theorem 2. *For any basic solution (y_{ij}, y'_{ij}) to LP2, if all variables for jobs uniquely assigned to a machine are fixed and at least one unfixed variable remains then at least one variable can be reduced relative to the remaining problem.*

Proof. Call (i,j) a *partial assignment* if $0 < y_{ij} + y'_{ij} < 1$. Consider the places of the partial assignments in the basis. There are two possibilities: either there exists a partial assignment not on a basis cycle, or all partial assignments are on the basis cycles.

Suppose there is at least one partial assignment not on the basis cycle. Consider a partial assignment (i, j) that is maximally distant (when measured in number of edges) from the basis cycle for its component. Note that i must be further from the cycle than j is and that i is otherwise incident to uniquely assigned nodes. Also, since i is not on the cycle, it must have no remaining capacity relative to the basic solution. Therefore, once the uniquely assigned jobs are fixed, the remaining capacity of the machine is exactly the flow on arc (i, j) times the size associated with (i, j). The first value is less than 1, since it is a partial assignment. The second is at most the maximum size of j on i. Therefore the resulting machine capacity is less than the maximum size of j on i.

Suppose all partial assignments are on the basis cycles. A basis cycle cannot satisfy condition 2 of Theorem 1 because a job at intermediate speed is either assigned completely to one machine (so no partial assignment is on the basis cycle) or to two or more machines (which gives a partial cycle not on the basis cycle). Therefore, all partial assignments must then be part of basis components that satisfy condition 3 of Theorem 1. Consider any component with a partial assignment on its basis cycle. If the flow on any edge of the basis cycle is 0 then remove that edge from the basis, reroot the resulting tree at any machine–node by treating the root node as having a degenerate cycle attached to it, and apply the previous argument. We can therefore assume that all edges on the basis cycle have nonzero flow.

Consider a machine node i on such a cycle. By Theorem 1, we know that i is scheduled to capacity. Let j and k be the job–nodes incident to i on the cycle and suppose the size corresponding to the basic variable between j and i (either u_{ij} if y'_{ij} is basic or l_{ij} if y_{ij} is basic) is no more than that between k and i. No other job incident to i can be a split job (because otherwise it would not be on the cycle, leading to the previous case). If the incident flow values sum to less than 1, then the new capacity of i will be strictly less than the maximum size of k (since it is a combination of two numbers no more than the maximum size of k, and the weights sum to less than 1). If the incident flow is more than 1, then there exists a node on the cycle with incident flow less than 1. Finally, if the incident flow equals 1 for all machine nodes around a cycle, then either there exists a machine node where the size of j is strictly less than the size of k or j and k have the same size for all machine nodes. In the former case, the new capacity of the machine is less than the maximum size of k. In the latter, the basis does not represent a linearly independent set. $\square$

This gives us a heuristic for this problem: fix all the values for jobs that are not split, reduce the variables, and resolve the linear relaxation. More formally, the heuristic is:

ALGORITHM LR–Heuristic

Input: Variable–speed problem, with n, m, b_i, l_{ij}, u_{ij}, c_{ij}, and s_{ij} as defined in the text.

Output: Heuristic solution y, y'.

Step 0) Reduce all variables. If no variable remains, then go to Step 3.

Step 1) Solve LP2 to get solution y, y'.

Step 2) For each job assigned to a unique machine, fix the corresponding variables, delete the job from the problem, and update the machine capacities. Go to step 0.

Step 3) Given the machine assignments, optimally schedule the individual machines using the single machine algorithm. Stop.

A naive analysis gives roughly mn executions of step 1 of LR–Heuristic. This can be decreased by the following observation.

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

Proof. We know from the previous theorem that a new variable gets reduced each iteration. In order for a variable to get reduced, it was necessary that at least one job be scheduled because otherwise the machine capacity is unaffected. After the first iteration, only m jobs remain to be scheduled. □

This approach has many advantages over other heuristics. First of all, the initial generalized network gives a lower bound on the solution, so there are bounds on the deviation from optimality. Second, the relaxation can be solved very effectively with the generalized network simplex method (Brown and McBride, 1985; Nulty and Trick, 1988), so a solution can be found very quickly. If polynomiality is required, the results of Khachian (1979) and Grötschel, Lovász and Schrijver (1988) suffice to show the polynomiality of solving LP2.

One disadvantage of this approach is that Theorem 2 requires fixing the size as well as the assignment of jobs. It would be better to only fix the assignment (the integer variable) and leave free the size (the continuous variable). It is not possible to ensure a reduction in this case, however. It is possible to check for what might be called a *strong reduction*: a reduction that occurs even if all variables take on their minimum size. In cases when a strong reduction prevents cycling, it is better to do only the strong reduction, fixing the assignments but not the sizes. Only when no strong reduction is available is the regular reduction done.

It is not possible to bound the deviation from optimality for this heuristic for a general cost function, nor could we for any polynomial heuristic unless $P = NP$ by Corollary 2. One important note, however, is that this heuristic may be unable to assign jobs to machines, even when the instance is feasible (this is not surprising, due to Corollary 1). Fortunately, we can bound the number of such jobs. From Corollary 4 it is clear that the difference between the optimal number scheduled and the number the heuristic schedules is no more than m. We can reduce this slightly.

Theorem 4. *For any instance where LP1 is feasible, LR–Heuristic fails to schedule no more than $m - 1$ jobs.*

Proof. It is possible to show that one variable doesn't become reduced, so at least one variable can be scheduled in the second iteration. See Trick (1991) for details. □

2.3. Violated Valid Inequalities. In the previous section, we showed how to use the linear relaxation solution to find good, heuristic, solutions. In this section, we show how to use the structure of the basis to improve the lower bound. In particular, we show how to generate a violated inequality directly from the relaxed solution.

The formulation in IP1 is a fixed cost network flow problem with lower and upper bounds. Van Roy and Wolsey (1986) have examined such problems and have devised some valid inequalities. We show that one of the classes they present must have a member that is violated by the linear relaxation. Furthermore, the violated member is easy to find. This gives us the first step in a cutting plane algorithm: we can solve the relaxation and add a first set of cuts. After adding those cuts, however, we cannot iterate, for the resulting problem does not have a generalized network basis structure.

Consider a machine i and a subset J of jobs, partitioned into J_1 and J_2 (either may be empty) such that

$$\sum_{j \in J_1} l_{ij} + \sum_{j \in J_2} u_{ij} = b_i + \lambda$$

with $\lambda > 0$. We call such a pair (j_1, j_2) a *generalized flow cover* (this is a specialization of the generalized flow cover of Van Roy and Wolsey). Let $(a)^+$ be a if $a \geq 0$ and 0 otherwise.

Theorem 5. *If (J_1, J_2) is a generalized flow cover for machine i, then the inequality*

$$\sum_{j \in J_2} (x_{ij} + (u_{ij} - \lambda)^+(1 - z_{ij})) + \sum_{j \in J_1} ((l_{ij} - \lambda)^+ + \min(l_{ij}, \lambda) z_{ij}) \leq b_i$$

is a valid inequality for the variable–speed scheduling problem.

Proof. This is Corollary 3 of Van Roy and Wolsey (1986) with $C_2 = \emptyset$. □

What makes this special case particularly interesting is that there is a constraint of this class for every reduced assignment found in Theorem 2.

Theorem 6. *For any basic solution of LP1 (LP2), if the solution is not feasible for IP1 (IP2) then there exists a violated inequality from the class of inequalities in Theorem 5.*

Proof. Consider a solution to LP1, with x_{ij} representing flow and the z_{ij} denoting the fraction of assignment (i, j) used. Note that we can generate this solution from the solution to LP2 by setting $z_{ij} = y_{ij} + y'_{ij}$ and $x_{ij} = l_{ij} y_{ij} + u_{ij} y'_{ij}$.

By Theorem 2, we know that there exists at least one variable (i, j^*) that can undergo variable reduction. Every variable identified by Theorem 2 has the following structure: the size of jobs uniquely assigned to the machine (all at either their minimum size or maximum size) plus either l_{ij^*} or u_{ij^*} (depending on which is basic) is more than b_i. To see this, suppose the variable to be reduced is not on a basis cycle. Then, by our choice, all other jobs assigned to the machine are non–split, and the result follows. If the variable to be reduced is on the cycle, then the machine may be incident to one other split–job, say k^*. But we chose to reduce the variable so that even if we decrease the value on (i, k^*) to 0, we are unable to increase the variable on (i, j^*) to 1. In either case, if you take J_1 to be all variables at their lower bounds (including j^* if y_{ij^*} is basic) and J_2 be those at their upper bounds (including j^* if y'_{ij^*} is basic), then (J_1, J_2) is a generalized flow cover.

Now, examine the valid inequality in Theorem 5 with respect to the linear relaxation solution. Suppose $j^* \in J_1$. Since for all $j \in J_1 \backslash j^* \cup J_2$, $z_{ij} = 1$, the left hand side reduces to:

$$\sum_{j \in J_2} x_{ij} + \sum_{j \in J_1 \backslash j^*} l_{ij} + (l_{ij^*} - \lambda)^+ + \min(\lambda, l_{ij^*}) z_{ij^*}.$$

Clearly $l_{ij^*} > \lambda$ since all of $J_1 \cup J_2 \backslash j^*$ fits on machine i but $J_1 \cup J_2$ does not. Therefore, the sum becomes:

$$\sum_{j \in J_2} u_{ij} + \sum_{j \in J_1 \backslash j^*} l_{ij} + (l_{ij^*} - \lambda) + \lambda z_{ij^*}$$

.

Note, however, that

$$\sum_{j \in J_2} u_{ij} + \sum_{j \in J_1 \backslash j^*} l_{ij} + l_{ij^*} - \lambda = b_i$$

by the definition of λ (this is true whether the partial assignment is on or off the basis cycle), and that our proof for Theorem 2 always reduces a variable with $y_{ij} + y'_{ij} > 0$. Therefore the left hand side has value more than b_i, as was required.

The case where $j^* \in J_2$ is similar. $\square$

Therefore, we can quickly find a valid inequality that the linear relaxation solution violates. We can then add this constraint to LP2 (doing the necessary variable substitutions) and resolve. Except in degenerate cases, this improves our lower bound.

The above theorem corresponds to regular reduction. There is a stronger constraint that corresponds to strong reduction when the corresponding variable is deleted (since the upper bound is less than the lower bound). In this case, we have identified a set of variables S such that even if all variables are at their lower bound, the total size is too large for the machine. Therefore we can generate the constraint

$$\sum_{j \in S} z_{ij} \leq |S| - 1$$

which will be violated and, in general, will be stronger than the constraints in Theorem 5. As stated in the previous section, however, strong reduction is not always available.

These constraints are not enough to create a cutting plane algorithm. Once we add the constraints, the generalized network structure of the problem is lost, so we can not repeat the process. It may be that the first round of cuts is sufficient to strengthen the lower bound, particularly if the cuts are lifted or otherwise strengthened (see Nemhauser and Wolsey (1988) for a survey on strengthening cuts).

3. The Makespan Model

We now turn to the case where we have uncapacitated machines and indivisible jobs and we wish to minimize the sum of the makespan and the operating costs. For the case in which there are operating costs but all the speeds are fixed, we create a polynomial time heuristic that is within 2.618 of optimal. For the case in which there are operating costs and variable speeds, we give a heuristic that is within $2.618 + \epsilon$ for any fixed epsilon. The heuristic is polynomial in the size of the problem and in ϵ. Although our latter computational results suggest that this heuristic is not likely to be useful in practice, it is important to notice that this heuristic shows that getting a constant bound is achievable. It was not until the work of Lenstra, Shmoys and Tardos (1990) that a constant bound was found for the case of fixed speed machines without operating costs. This result extends their result.

The problem we wish to solve is IP3.

(IP3)

Minimize $k + \sum_i \sum_j c_{ij} z_{ij} + \sum_i \sum_j s_{ij}(u_{ij} z_{ij} - x_{ij})$

Subject to

$\sum_j x_{ij} \leq k$ for all i,

$\sum_i z_{ij} = 1$ for all j

$l_{ij} z_{ij} \leq x_{ij} \leq u_{ij} z_{ij}$ for all i, j,

$x_{ij} \geq 0$ for all i, j,

$z_{ij} \geq 0$ and integer for all i, j.

Let LP3 denote the linear relaxation of this problem.

First note that if we fix k then we get a problem in the form of IP1, which we denote

IP1(k). Also note that without altering the optimal integer solution to IP1(k), we can reduce the maximum time allowed for a job on a machine down to the value k. Furthermore, if this value is less than the minimum amount of time for that job on that machine, we can delete that assignment completely. Call this process *reduction* (note that this process is identical to the reduction process in section 2). Let LP1(k) denote the linear relaxation to IP1(k) after reduction. Let $f(k)$ be the value of optimal solution to LP1(k). If the linear relaxation has no feasible solution, set $f(k) = \infty$.

There are two steps in creating a heuristic solution within a constant factor of optimality: first we need to find a k such that $f(k) + k$ is either a lower bound on the optimal value of IP3 or is no more than a constant factor above the value of IP3. Then we need to find a feasible integer solution that does not increase the objective by more than a constant factor. The following theorem handles the second of these tasks.

The fundamental tool we use is called *δ–roundoff.* Consider any basic feasible solution to LP1(k) and any $0 \leq \delta \leq 1$. Give each basis cycle an arbitrary orientation. For any job j, we define the *associated machine* for j to be the unique machine incident (in the basis graph) to the job that is closer to the basis cycle, or, if the job is itself on the basis cycle, the machine incident by the arbitrary orientation of the basis cycle. The set of all other machines incident to j in the basis graph are the *auxiliary machines.* Note that every machine is the auxiliary machine for at most one job. δ–roundoff is defined as follows:

For each job j, if j has more than δ assigned to any machine, then completely assign j to that machine. This is called a *large* assignment. Otherwise, completely assign j to its cheapest cost auxiliary machine (giving a *small* assignment).

Theorem 7. *For any basic feasible solution to LP1(k^*) with objective value* $f(k^*)$ *and any* $0 < \delta < 1$, *it is possible to find in polynomial time a solution to IP3 with objective no more than* $\max\{1/\delta, 1/(1-\delta)\}f(k^*) + (1 + 1/\delta)k^*$.

Proof. We prove this by analyzing the effect of large and small assignments separately.

Consider the effect of a large assignment. The size of that job increases by a factor of at most $1/\delta$. The cost increases by the same factor. Therefore, the total effect of all large assignments is an increase in cost and makespan of a factor of $1/\delta$. During a small assignment, the cost increases by a factor of at most $1/(1-\delta)$. The size may go up (absolutely) by as much as k^*, since no arc has size more than that. But each machine is only assigned one small assignment since each machine is auxiliary to at most one job, so the effect of all small assignments is to increase cost by a factor of no more than $1/(1-\delta)$ and to increase the makespan by at most k^*. Therefore, total size on any machine is at most $(1 + 1/\delta)k^*$ and the cost has increased by a factor of no more than $\max\{1/\delta, 1/(1-\delta)\}$.

This analysis is summarized in Table I. All values are the factor increase, except the increase in makespan for a small assignment which is an additive value.

Type of Assignment	Increase in Makespan	Increase in Cost
Large	$1/\delta$	$1/\delta$
Small	$+k^*$	$1/(1-\delta)$

TABLE I. Summary of Analysis

□

Let $\phi = (1+\sqrt{5})/2 \approx 1.618$. The following corollary shows that an appropriate choice of δ results in a heuristic with a worst case bound of 2.618 of optimal.[1]

Corollary 5. *For any basic solution to LP1(k^*) with objective value $f(k^*)$, it is possible to find in polynomial time a solution to IP3 with objective no more than $(1+\phi)(k^* + f(k^*)) \approx 2.618(k^* + f(k^*))$.*

Proof. The maximum value for the bound in Theorem 7 can be found by setting the makespan increase equal to the cost increase. If we assume $\delta \geq 1/2$, then this is equivalent to setting $1/\delta = 1 + 1/\delta$. Setting $\delta = (\phi - 1)$ gives the required bound. Setting $\delta < 1/2$ results in worse bounds. □

The following result is shown in Lenstra, Shmoys and Tardos (1990). This corollary shows that their result is a special case of Theorem 7.

Corollary 6. *If there are no operating costs, then for any basic solution to LP1(k^*) it is possible to find in polynomial time a solution to IP3 with objective no more than $2k^*$.*

Proof. In the proof of Theorem 7, it is permissible to set $\delta = 1$ if $f(k^*) = 0$. This then gives a bound of $2k^*$. □

This analysis is tight, in the sense that for any δ there exist instances where δ–roundoff increases the objective by the amount given in Theorem 7.

The next step is to determine a suitable k. If we did not reduce any processing times, then $f(k)$ would be convex, and we could use standard nonlinear programming

[1]Shmoys and Tardos (1992) have recently improved this bound to $2k^* + f(k^*)$ by a clever rounding mechanism involving solving a matching problem on a related graph. Note that their result is stronger in that while the makespan at most doubles, the cost is not increased at all.

techniques to minimize $f(k) + k$ to get a true lower bound. Since we do reduce the processing times, however, (and need to in order to get a constant bound) we need to use another method.

Our method is to divide the range of k up into intervals where the feasible arcs remain constant. We then solve the problem within each interval and take the best of those solutions.

Note that if u^* denotes the maximum time any job can take on any machine, then nu^* is an upper bound on the makespan in any optimal solution. Also, we can assume that all of the maximum and minimum times on a machine are integer by scaling the problem if necessary. In this case, we can prove the following result.

Lemma 3. *There exists an optimal solution to IP3 with integral $k \le nu^*$.*

Proof. The only difficulty is in proving k integral. We prove this claim by showing that there exists an optimal solution to IP3 such that there is a machine i with every job assigned to that machine using either its minimum or its maximum time. Let (x', z', k') represent an optimal solution to IP3. Consider the set S of machines used to capacity by this solution. If any machine in S does not contain an intermediate job (one with $l_{ij} < x_{ij} < u_{ij}$), then k' must be integral. By the results Lemma 1, there exists at most one job on a machine at intermediate time, so each machine in S has exactly one intermediate job. For each $i \in S$, let $j(i)$ denote the intermediate job on i.

Let $\epsilon = \min_{i \in S}\{u_{ij(i)} - x_{ij(i)}\}$ and note that $\epsilon > 0$. Consider increasing k' and each of the corresponding intermediate $x_{ij(i)}$ for all $i \in S$ by ϵ. By the definition of ϵ, this gives a feasible solution, and the objective changes by some value $\Delta\epsilon$, which must be non–negative by the optimality of (x', z', k'). Now consider decreasing k' and each of the $x_{ij(i)}$, $i \in S$ by a small amount δ. Since all machines not in S are not used to capacity, there must be a suitably small δ so that the resulting solution is feasible. The objective now changes by $-\Delta\delta$, which also must be nonnegative. Therefore, $\Delta = 0$. Therefore, increasing k' by ϵ results in an alternative optimal solution where one machine used to capacity has no intermediate jobs. This alternative optimum has integral makespan. $\square$

Let us begin with the case where, for each job and each machine, the maximum time for the job on the machine equals the minimum time. In this case, if we reduce the processing times for any job, we delete the assignment completely. In this case, we can find a k such that $f(k) + k$ is a lower bound on the value of IP3.

The key is to divide the feasible region for k into intervals where the set of feasible assignments is constant. This is done in Algorithm Fixed–Find–k.

ALGORITHM Fixed–Find–k

Input: An instance of the makespan model, IP3, with $l_{ij} = u_{ij}$ for all i and j.

Output: A value k^* such that $f(k^*) + k^*$ is a lower bound on the optimal value for IP3.

```
Step 0) $k^* = 0$.
Step 1) For each $1 \leq i \leq m$ and $1 \leq j \leq n$ do
    Add the constraint $k \geq l_{ij}$ to LP1($l_{ij}$)
    Solve to get optimal $k$ value, $k'$
    If $f(k') + k' < f(k^*) + k^*$, set $k^* = k'$.
Step 2) Stop.
```

Lemma 4. *Algorithm Fixed–Find–k returns a k^* such that $f(k^*) + k^*$ is a lower bound on the optimal solution value to IP3.*

Proof. Consider any optimal solution (x', z', k') to IP3, and let ℓ' denote the largest value for l_{ij} with $z'_{ij} = 1$. Now consider LP1(ℓ'). Note that all arcs with positive z' represent feasible arcs in LP1(ℓ'). Furthermore, $k' \geq \ell'$, since the makespan cannot be any less than the length of any assignment used. Therefore, (x', z', k') is a feasible solution to LP1(ℓ'), so the optimal solution to that problem must be a lower bound. This implies that the minimum over all LP1(l_{ij}) must also be a lower bound. $\square$

Theorem 8. *There exists a polynomial time heuristic for the fixed processing time problem with worst case bound* $1 + \phi \approx 2.618$.

Proof. By Lemma 4, it is possible to find a k^* for which $f(k^*) + k^*$ is a lower bound on the optimal solution to IP3. To do so, $O(mn)$ linear programs must be solved. By the results of Khachian (1979) and Grötschel, Lovász and Schrijver (1988), it is possible to find an optimal basic solution to a linear program in polynomial time, so we can find k^* in polynomial time.

Using Lemma 7, we then get the desired bound. $\square$

We can now address the case of general processing times. The difficulty with using the above argument is that maximum times on a machine must also be reduced, but it is impossible to know by how much until the makespan is determined. To solve this problem, we will find a solution that is not a lower bound itself, but is not too much bigger than the optimal solution.

Fix a value $\alpha > 1$. The following algorithm will find a solution that is no more than a factor of α larger than the optimum.

ALGORITHM Variable–Find–k

Input: An instance of the makespan model, IP3. A fixed value $\alpha > 1$.

Output: A value k^* such that $(f(k^*) + k^*)/\alpha$ is a lower bound on the optimal value for IP3.

```
Step 0) $k^* = 0$.
Step 1) For each integral $p$ such that $0 \leq \alpha^p \leq \alpha n u^*$
    If $f(\alpha^p) + \alpha^p < f(k^*) + k^*$, set $k^* = \alpha^p$.
Step 2) Stop.
```

Lemma 5. *Algorithm Variable–Find–k returns a k^* such that $(f(k^*) + k^*)/\alpha$ is a lower bound on the optimal solution value to IP3.*

Proof. Let (x', z', k') be an optimal solution to IP3, and let p be such that $\alpha^{p-1} < k' \leq \alpha^p$. By Lemma 3 such a p exits. Let the cost for the optimal solution be split into k' (the makespan) and C' (the operating costs).

Now consider $f(\alpha^p) + \alpha^p$. Since $\alpha^p \geq k'$, $C' \geq f(\alpha^p)$. Also, by our choice of p, $\alpha^p \leq \alpha k'$. Combining these we see that $f(\alpha^p) + \alpha^p \leq C' + \alpha k' \leq \alpha C' + \alpha k'$. This implies that k^* gives a solution within a factor of α, as required. $\square$

Theorem 9. *For any $\alpha > 1$, there exists a polynomial time heuristic for the variable speed problem with a worst case factor of $\alpha(1 + \phi) \approx \alpha 2.618$.*

Proof. The number of feasible p in step 1 of Variable–Find–k is $O(\log_\alpha nu^*)$. For fixed α, the base to which the logarithm is taken does not affect the overall complexity. Therefore, a polynomial number of linear programs must be solved.

By Theorem 7, we can round the solution for k^*, increasing its value by at most a factor of $1+(1+\sqrt{5})/2$. This gives a total bound of $\alpha(1+(1+\sqrt{5})/2)$, as needed. $\square$

This gives a worst case ratio limit of $1 + \phi \approx 2.618$. Lenstra, Shmoys, and Tardos show that if there are no operating costs and no choice in the operating speed then no polynomial algorithm can have a worst case bound of 1.5 (unless $P = NP$). This bound naturally holds for this more general problem, leaving a gap between 2.618 and 1.5 as the best possible approximation algorithm.

4. Computational Results

In this section, we present computational results for both the capacitated machine model and the makespan model. Our heuristics are fast, robust, and accurate.

4.1. Capacitated Machines.

4.1.1. *Other Heuristics.* Solving the linear relaxation is only one heuristic for the multiple capacitated machine problem. In this section, we will outline two other heuristics that form the base of our computational tests.

Because we know an optimal method for single machines, we can simply search for assignments of jobs to machines. One we have an assignment of jobs to machines, we can apply our single machine algorithm to determine the optimal speeds. The difficulty is in the assignment of jobs to machines.

The first heuristic is a greedy heuristic. Assume we have an arbitrary ordering of the jobs (here denoted $1, 2, \ldots, n$, but in general a permutation of this set). We divide the problem into n stages, where the subproblem to solved at stage j is the assignment of the jth job, given the assignment of jobs 1 through $j - 1$.

We solve the subproblem in the most straightforward way: during stage j, we determine the cost of assigning j to each machine i in turn. We assume the assignments (though not the sizes) of 1 up to $j-1$ are fixed, and no job after j is considered. The job is then assigned to the machine that gives the minimum total cost.

Clearly, this is a very simple–minded heuristic. Jobs that appear early in the sequence ignore those that appear later, so assignments are made as though the machine is not filled to capacity. Jobs that appear late in the sequence might have to use very expensive machines in order to satisfy feasibility. One advantage of this heuristic, however, is that different orderings of the jobs lead to different solutions. This heuristic is good for generating many different solutions.

Our second heuristic attempts to handle the problem of jobs being left with only one expensive machine. Here there is no ordering of jobs. The jobs are assigned to machines one at a time. The job to be assigned is chosen by determining the maximum *regret* for not being able to use its best machine. The regret is defined to be the difference in costs between assigning it to its best machine and assigning it to its second best machine. The costs are determined by assuming that all previously assigned jobs are fixed and all unassigned jobs are irrelevant.

This heuristic still ignores the congestion that results from later assignments, but it does try to identify jobs that have just one attractive machine available. The solution found by this heuristic is termed the *savings–regret* solution.

Once we have a solution, there are a number of ways we might improve on it. For instance, job j might be assigned to machine i but moving it to machine i' decreases the total cost. Such a move is called a 1–exchange. Or it may be that switching j and j' decreases cost. Such a switch is a 2–exchange. We can generalize this to k–exchanges for arbitrary k, but 2–exchanges are sufficient for our purposes. Given a solution, we can do 1– and 2–exchanges until no improvement is possible. This gives a *locally optimal solution.*

4.1.2. *Computational Results.* In this section we present some computational results which show how well the heuristics and lower bound techniques work. In general, LR–Heuristic works very well on the problems we generate, but the violated valid inequalities do not affect the lower bound much.

We generate random problems using a generator similar to that used in a number of papers (Ross and Soland, 1975; Ross and Soland, 1977; Martello and Toth, 1981; Fisher, Jaikumar and Van Wassenhove, 1986; Jacobs, 1987) for the generalized assignment problem (generators A and B). We choose a number of jobs and number of machines. For each assignment, a fixed cost is randomly generated uniformly between 15 and 50. A minimum size is generated uniformly between 5 and 25. We also generate a variable profit between 0 and 5, and a maximum size, generated between 0 and some given range above the minimum size. Finally we have an average target size, and the capacity of each machine is set to the target size times the number of jobs divided by the number of machines. All data are integer.

Our first test compares three heuristics: greedy, savings–regret, and LR–Heuristic, the heuristic based on the linear relaxation. For this test a target size of 15 (the average minimum size on a machine) is fixed. The range of maximum sizes above the minimum size is either 3 (narrow range), 10 (medium range), or 20 (wide range). We test four sizes of problems: 20 jobs, 5 machines; 50 jobs, 5 machines; 50 jobs, 10 machines; 100 jobs, 10 machines; and 200 jobs, 10 machines. Ten problems were run for each upper bound range. The solutions for each heuristic were improved by 1– and 2–exchanges.

Table II compares the solution quality generated by our heuristics both with and

Table II. Percentage over Lower Bound

		Without Improvement			With Improvement		
Jobs,Mach.	Range	Greedy	Savings–Reg	LR–Heur	Greedy	Savings-Reg	LR–Heur
	3	15.4	40.2	15.4	3.9	5.1	2.6
20,5	10	38.6	31.0	29.8	7.0	7.1	4.5
	20	56.1	50.7	40.9	9.7	13.4	8.9
	3	6.8	7.6	6.1	1.3	1.5	0.0
50,5	10	36.4	37.9	14.7	5.9	5.5	4.3
	20	46.1	43.2	32.9	9.8	10.1	8.1
	3	13.9	8.7	8.0	5.0	2.5	1.2
50,10	10	25.7	24.5	70.0	6.3	7.6	5.3
	20	51.3	43.3	95.4	10.3	10.9	8.4
	3	7.9	6.9	7.9	1.7	1.6	0.5
100,10	10	28.3	23.9	28.2	5.9	5.9	4.1
	20	51.2	52.0	62.6	10.9	10.0	8.8
	3	6.4	5.3	3.8	1.0	1.0	0.4
200,10	10	32.6	29.9	15.8	4.6	5.1	3.7
	20	56.5	55.7	35.8	9.9	10.4	8.8

—Figure 2 around here—

FIGURE 2. Comparison of heuristics

without the improvement heuristics. Each entry represents the average percentage above the lower bound (as generated by the linear relaxation). Clearly the heuristics based on the linear relaxation together with the improvement heuristics do a good job. In every instance type, this combination resulted in the best values. Furthermore, this average performance is consistent across instances: of the 150 instances represented in the table, LR–Heuristic with improvement had the best solution in 132 cases.

Figure 2 represents another way to compare the results. To make the numbers roughly comparable across categories, the results are presented as a percentage of the gap between the lower bound and the greedy solution (without the exchange heuristics). In other words, if the raw greedy value was 50, and the lower bound was 25, a heuristic of value 30 would get value 20% (equals (30-25)/(50-25) * 100). This measure, while slightly arcane, is independent of various data transformations that keep the problem essentially the same. For instance, percentage above lower bound changes if a constant is added to all the costs. The value used does not. Another way to think about this measure is the following: suppose the current method for solving this problem is to use the greedy heuristic without exchanging. This gives a gap above the lower bound. The entry in the figure gives, for each size and degree of size range, the percentage of gap remaining if the raw greedy heuristic is replaced with the alternative heuristics (with exchanging).

Furthermore, while it is clear that solving generalized networks is a time consuming task, the time required to do the exchanging is considerable. The average times for each problem size (size range had no noticeable effect on computation time) are given

TABLE III. Computation Time (sec.)

	Without Improvement			With Improvement		
Jobs,Machines	Greedy	Savings–Reg	LR–Heur	Greedy	Savings-Reg	LR–Heur
20,5	0.0	0.0	0.6	0.0	0.2	0.8
50,5	0.0	1.2	1.0	2.9	3.2	2.5
50,10	0.3	2.5	1.9	3.2	4.3	4.1
100,10	0.5	14.2	4.1	17.2	36.3	17.8
200,10	0.9	106.3	10.3	205.2	221.7	149.1

in table III. Times are in seconds on a Sun SPARC workstation.

The data structure used to store a solution consisted of a doubly linked list with the jobs sorted in order of their variable cost. Despite this, it takes a tremendous amount of time to do a 2–exchange. This is reflected in the high times for greedy and savings–regret above. LR–Heur finds much better initial solutions so its overall computation time is not as large.

The final test determines the effectiveness of the lower bounds generated. Cuts were generated for all of the instances were added to the problem. The problem was resolved using a linear programming package. In practice, a generalized network with side constraints code (McBride, 1985) would be more efficient. Table IV gives the results. Again, the entries in the table are given in terms of the percentage over the lower bound. In order to aid in comparisons, the values for LR–Heuristic with improvement are repeated in this table. It is clear that the lower bound is not increased a lot.

Table IV also gives a comparison with the optimum integer values. For both of the two smallest problem sizes, the optimum solution was found using branch and

TABLE IV. Cuts and Optimal (Percentage above lower bound)

Jobs,Machines	Range	LR–Heur	Cuts	Optimum
	3	2.6	0.6	2.0
20,5	10	4.5	0.4	2.6
	20	8.9	0.9	3.2
	3	0.5	0.1	0.3
50,5	10	4.3	0.1	0.5
	20	8.1	0.1	0.7
	3	1.2	0.2	—
50,10	10	5.3	0.3	—
	20	8.4	0.3	—
	3	0.5	0.0	—
100,10	10	4.1	0.1	—
	20	8.8	0.1	—
	3	0.4	0.0	—
200,10	10	3.7	0.0	—
	20	8.8	0.0	—

bound. It seems that both the heuristic solution and the lower bounding techniques have room for further improvement.

4.2. Makespan and Operating Costs model. Our second major heuristic was a guaranteed accuracy round off heuristic to minimize the sum of makespan and operating costs. A natural question concerns the nature of the worst case bound we get. Does the heuristic typically achieve the worst case or is the worst case pathological in some sense?

To get a feel for the answer to this question, we have modified the generator from the previous section to create instances of the makespan model. Jobs characteristics are generated in exactly the same way as before. Rather than generating a machine size, however, there is a a new parameter *weight* that gives the relative weight to be assigned to makespan compared to operating costs. The objective value for makespan is normalized to one; operating costs are then divided by the weight.

Problems were generated for the five problem sizes of the previous section. The size range was fixed at the medium value 10. We tested three weight values: 1, 10, and 50. In the routine Variable–Find–k, we set $\alpha = 2$ so we have a worst case ratio 5.236. For each size and weight combination, ten instances were generated. After finding the appropriate k value, two feasible solutions were generated: the first uses the round–off rule given in Section 3; the second solution then fixes the makespan and applies the 1– and 2– opting improvement techniques of the previous section. Again, for the two smallest sizes, the optimal integer solution was found using branch and bound. The results are given in Table V.

From the table, it is clear that the worst case ratio is not a good estimate on how well this heuristic will work in practice; in general the heuristic works much better.

TABLE V. Empirical Ratios for Makespan Plus Operating Costs

Jobs,Machines	Weight	Round-Off	Improvement	Optimal
	1	1.044	1.029	1.011
20,5	10	1.224	1.204	1.025
	50	1.584	1.572	1.037
	1	1.079	1.060	1.002
50,5	10	1.089	1.072	1.003
	50	1.518	1.512	1.001
	1	1.049	1.041	—
50,10	10	1.213	1.176	—
	50	1.493	1.450	—
	1	1.046	1.035	—
100,10	10	1.082	1.059	—
	50	1.255	1.214	—
	1	1.043	1.038	—
200,10	10	1.052	1.042	—
	50	1.177	1.146	—

As makespan receives more weight, the quality of the heuristic decreases. To further explore this, more instances were generated for the 50 job, 5 machine problem and many more weights were generated. This is illustrated in Figure 3. Again, each data point represents the average of ten instances. It seems as though the quality of this heuristic is quite good as long as the makespan weight does not dominate the operating cost. Once the makespan weight is too large, however, then the heuristic is consistently off by more than 70%.

—Figure 3 around here—

FIGURE 3. Solution Quality vs. Makespan Weight

5. CONCLUSIONS

We have examined an interesting generalization of standard multiple machine scheduling: variable–speed scheduling. We have shown that the linear relaxation solution contains a lot of information. Most jobs are assigned to only one machine, leaving just a small number to be rescheduled. Furthermore, it is possible to use the linear relaxation to reschedule the rest. Also, it is possible to generate violated valid inequalities from the optimal basis.

Limited computational tests suggest that the resulting heuristic gives good answers consistently, and in a reasonable amount of time. Adding the violated constraints does not increase the lower bound much.

There are a number of other questions to answer for this problem and related problems. First, it is clear that more testing is needed to determine the quality of the heuristic. More sophisticated competitors can be devised.

Second, it is possible to combine the heuristic and the constraint generation in the following iterative algorithm:

(Lagrangian Heuristic)

Step 0) Solve LP2.

Step 1) Find LR–Heuristic solution from current basis.

Step 2) Find violated valid inequality(ies) from current basis and add them to current set of inequalities.

Step 3) Relax current set of inequalities by lagrangian relaxation (Fisher, 1985), and go to step 1.

The loop can terminate with any standard terminating condition: time limits, iteration limits, convergence, and so on.

Step 1 generates a series of feasible solutions; step 3 generates a series of lower bounds. In general, due to the limited nature of the cuts we generate, we will soon cycle. This is not too critical because solving a lagrangian relaxation each iteration is a very expensive process so we do not wish to do many iterations. Ideally, however, the (non–decreasing) lower bounds and the generation of many feasible solutions may substantially decrease the duality gap.

Third, the violated valid inequalities are generated from one mixed integer constraint. It would be interesting to come up with other valid inequalities that come from more than one constraint. Gottlieb and Rao (1990b)(1990a) have done this for the generalized assignment problem; the variable–speed scheduling problem is a natural generalization.

Finally, this whole approach suggests that integer generalized networks are an interesting class of mixed integer programs. The network structure leads very naturally to interesting relaxations and heuristics. Furthermore, there is enough structure to make finding violated inequalities easier than for general mixed integer programming.

References

Brown, G. G. and R. McBride 1985. Solving Generalized Networks. *Management Science*, **30**, 1497–1523.

Daniels, R. L. and R.K. Sarin 1989. Single machine scheduling with controllable processing times and number of jobs tardy. *Operations Research*, **37**, 981–984.

Fisher, M. 1985. An applications oriented guide to lagrangian relaxation. *Interfaces*, **15**, 10–21.

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

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

Gottlieb, E. and M.R. Rao 1990b. The generalized assignment problem: valid inequalities and facets. *Mathematical Programming*, **46**, 31–52.

Gray, A., A. Seidmann, and K.E. Stecke 1988. Tool management in automated manufacturing: operational issues and decision problems. Technical report, Center for Manufacturing and Operations Management, University of Rochester.

Grötschel, M., L. Lovász, and A. Schrijver 1988. *Geometric Algorithms and Combinatorial Optimization.* Berlin: Springer.

Jacobs, C. D. 1987. *The Vehicle Routing Problem with Backhauls.* PhD thesis, Georgia Institute of Technology.

Khachian, L. 1979. A polynomial time algorithm in linear programming. *Soviet Mathematics Doklady*, **20**, 191–194.

LENSTRA, J., D.B. SHMOYS, AND E. TARDOS 1990. Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, **46**, 259–271.

MALAKOOTI, B. AND J. DEVIPRASAD 1989. An interactive multiple criteria approach for parameter selection in metal cutting. *Operations Research*, **37**, 805–818.

MARTELLO, S. AND P. TOTH 1981. An algorithm for the generalized assignment problem. In J. Brams (Ed.), *Operational Research '81.* New York: North Holland.

MCBRIDE, R. 1985. Solving embedded generalized network problems. *European Journal of Operational Research*, **21**, 82–92.

NEMHAUSER, G. L. AND L. A. WOLSEY 1988. *Integer and Combinatorial Optimization.* New York: John Wiley.

NULTY, W. G. AND M.A. TRICK 1988. GNO/PC generalized network optimization system. *O.R. Letters*, **2**, 101–102.

PHILLIPSON, R. AND A. RAVINDRAN 1979. Applications of mathematical programming to metal cutting. *Mathematical Programming*, **23**, 1001–1023.

POTTS, C. 1985. Analysis of a linear programming heuristic for scheduling unrelated parallel machines. *Discrete Applied Mathematics*, **10**, 155–164.

ROSS, G. AND R.M. SOLAND 1975. A branch and bound algorithm for the generalized assignment problem. *Mathematical Programming*, **8**, 91–103.

ROSS, G. AND R.M. SOLAND 1977. Modelling facility location problems as generalized assignment problems. *Management Science*, **24**, 345–357.

SCHWEITZER, P. AND A. SEIDMANN 1989. Performance optimization and capacity range analysis for FMS's with distinct multiple job visits to work centers. Technical Report QM8910, William E. Simon Graduate School of Business Administration, University of Rochester.

Schweitzer, P. and A. Seidmann 1991. Processing rates optimization in flexible manufacturing systems. *Management Science*, **37**, 454–466.

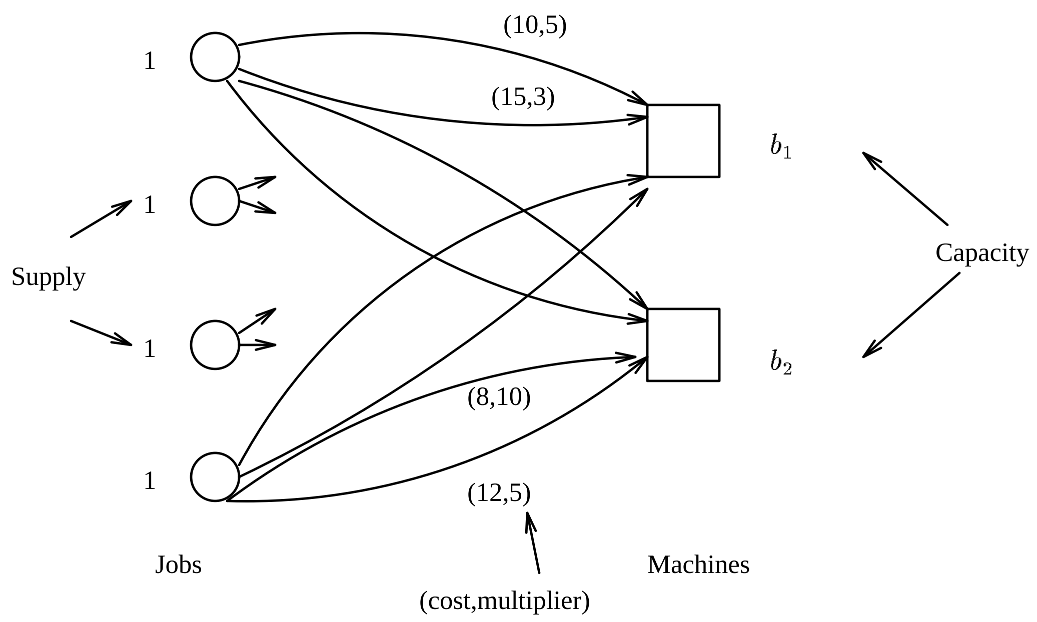
Shmoys, D. and E. Tardos 1992.

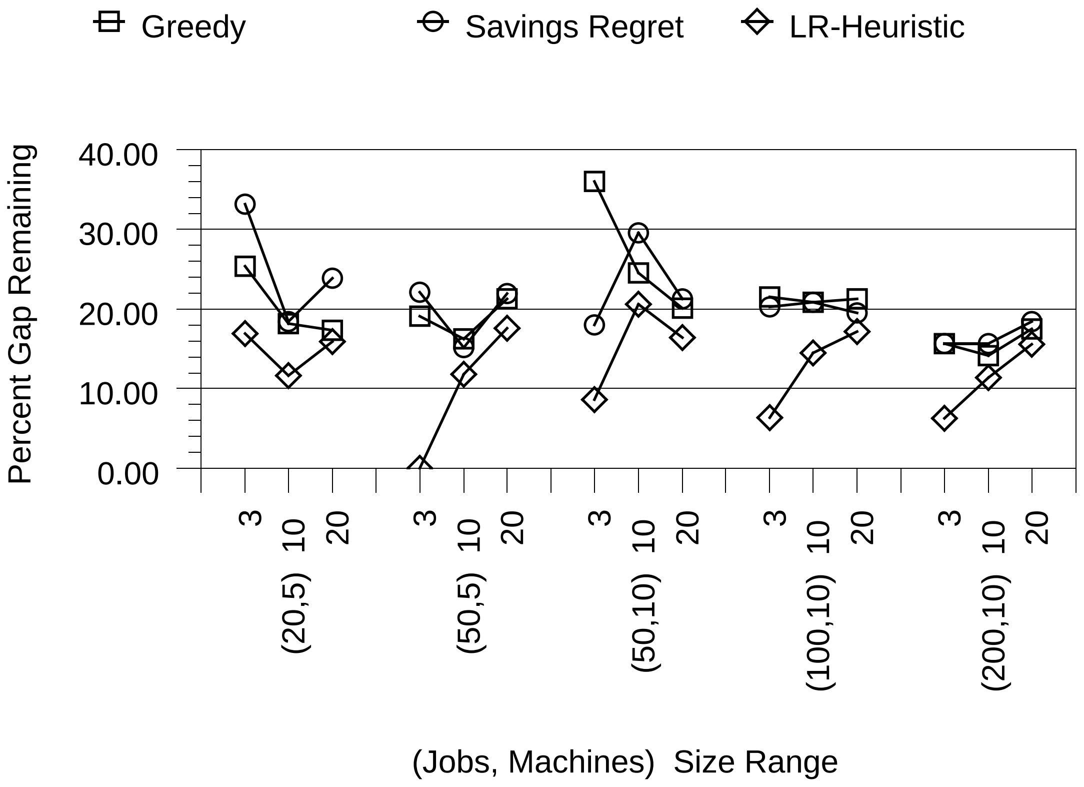
Trick, M. A. 1991. A linear relaxation heuristic for the generalized assignment problem. *Naval Research Logistics*, **39**, 137–152.

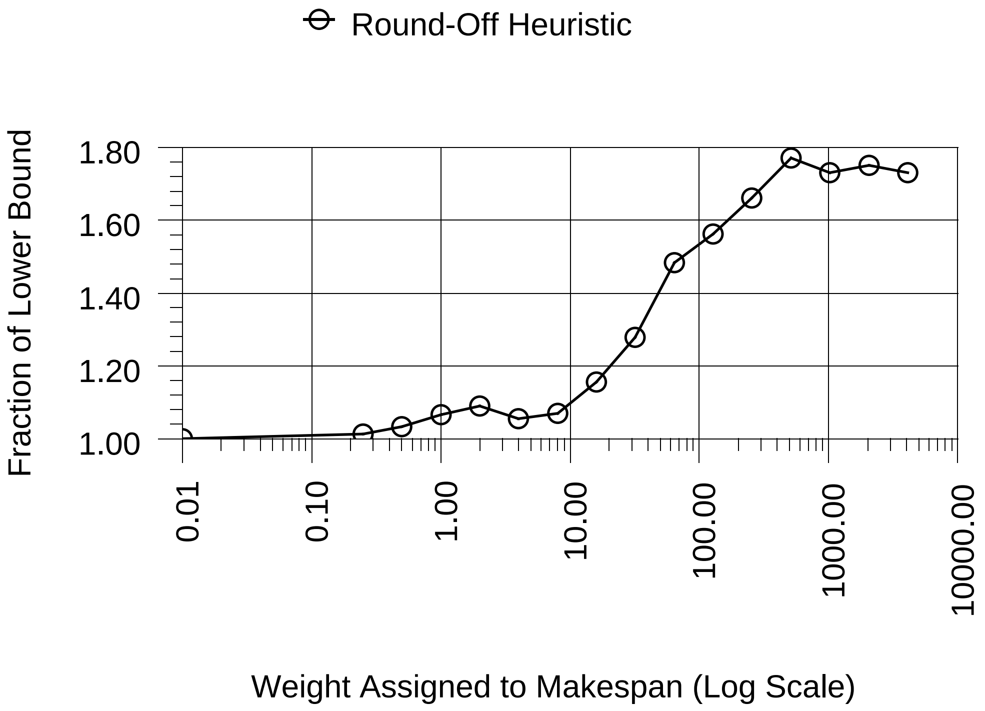
Van Roy, T. and L.A. Wolsey 1986. Valic inequalities for mixed 0–1 programs. *Discrete Applied Mathematics*, **14**, 199–213.

Vickson, R. 1980a. Choosing the job sequence and processing times to minimize total processing plus flow cost on a single machine. *Operations Research*, **28**, 1155–1167.

Vickson, R. 1980b. Two single–machine sequencing problems involving controllable job processing times. *AIIE Transactions*, **12**, 258–262.


(10,5)
(15,3)
1
1
1
1
Supply
b_1
b_2
Capacity
(8,10)
(12,5)
Jobs
Machines
(cost,multiplier)


Greedy
Savings Regret
LR-Heuristic
Percent Gap Remaining
40.00
30.00
20.00
10.00
0.00
3
10
20
(20,5)
(50,5)
(50,10)
(100,10)
(200,10)
(Jobs, Machines) Size Range


Round-Off Heuristic
Fraction of Lower Bound
1.80
1.60
1.40
1.20
1.00
0.01
0.10
1.00
10.00
100.00
1000.00
10000.00
Weight Assigned to Makespan (Log Scale)