ORSA Journal on Computing
Vol. 3, No. 2, Spring 1991

0899-1499/91/0302-0149 $01.25

A Computational Study of the Job-Shop Scheduling Problem

DAVID APPLEGATE *School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, ARPANET: david@theory.cs.cmu.edu*

WILLIAM COOK *Combinatorics and Optimization Research Group, Bellcore, P.O. Box 1910, Morristown, NJ 07960-1919, ARPANET: bico@bellcore.com*

(Received: August 1990; final revision received: December 1990; accepted: January 1991)

The job-shop scheduling problem is a notoriously difficult problem in combinatorial optimization. Although even modest sized instances remain computationally intractable, a number of important algorithmic advances have been made in recent years by J. Adams, E. Balas and D. Zawack; J. Carlier and E. Pinson; B. J. Lageweg, J. K. Lenstra and A. H. G. Rinnooy Kan; and others. Making use of a number of these advances, we have designed and implemented a new heuristic procedure for finding schedules, a cutting-plane method for obtaining lower bounds, and a combinatorial branch and bound algorithm. Our optimization procedure, combining the heuristic method and the combinatorial branch and bound algorithm, solved the well-known 10 × 10 problem of J. F. Muth and G. L. Thomson in under 7 minutes of computation time on a Sun Sparcstation 1.

The *job-shop problem* is to schedule a set of jobs on a set of machines, subject to the constraint that each machine can handle at most one job at a time and the fact that each job has a specified processing order through the machines. The objective is to schedule the jobs so as to minimize the maximum of their completion times.

This problem is not only NP-hard[17] it also has the well-earned reputation of being one of the most computationally stubborn combinatorial problems considered to date. This intractability is one of the reasons the problem has been so widely studied. Indeed, some of the excitement in working on the problem no doubt arose from the fact that a specific instance, with 10 machines and 10 jobs, posed in a book by Muth and Thompson[19] in 1963 remained unsolved for over 20 years. This particular instance was finally settled in 1985 by Carlier and Pinson,[10] culminating a steady stream of algorithmic improvements over the years, by Ashour and Hiremath,[2] Balas,[3] Barker,[6] Barker and McMahon,[7] Bratley, Florian and Robillard,[8] Fisher, Lageweg, Lenstra and Rinnooy Kan,[12] Florian, Trépant and McMahon,[13] Lageweg, Lenstra and Rinnooy Kan,[14] Lenstra,[16] Rinnooy Kan,[22] and others.

The purpose of our paper is twofold. On the one hand, we would like to demonstrate the effectiveness of the procedures we have developed. On the other hand, we point out a list of 7 open problems from the literature that we were unable to solve and which may be candidates to take the place of the now solved problem of Muth and Thompson.

The paper is organized as follows. In Section 1, we describe a cutting-plane method for obtaining lower bounds on job-shop problems. This work makes use of valid inequalities given by Balas,[5] Dyer and Wolsey,[11] McGee,[18] and ourselves. In Section 2 we present a combinatorial branch and bound algorithm that incorporates the key ideas of Carlier and Pinson[10] with a number of enhancements. Section 3 contains a new heuristic procedure for finding schedules, combining the nice heuristic method of Adams, Balas, and Zawack[1] and the kernel of the branch and bound method described in Section 2. Finally, in Section 4 we report on the status of a group of test problems described in [1] and [15].

1. Cutting Planes

We begin by specifying the job-shop problem in a more precise manner. As input, we have a finite set J of *jobs* and a finite set M of *machines*. For each job $j \in J$ we are given a permutation $(\sigma_1^j, \ldots, \sigma_m^j)$ of the machines (where $m = |M|$), which represents the processing order of j through the machines. Thus j must be processed first on σ_1^j, then on σ_2^j, etc. Also, for each job j and machine α we are given a nonnegative integer $p_{j\alpha}$, the processing time of j on α. The objective is to find a schedule of J on M that minimizes the maximum of the completion times of the jobs J.

Subject classification: Production/scheduling: deterministic, multiple machine.

For each job j and each machine α, let $x_{j\alpha}$ be the starting time of j on α. The constraints on the vector $(x_{j\alpha}: j \in J, \alpha \in M)$ to represent a schedule are as follows:

$$x_{j\alpha} \geq 0 \text{ for all } j \in J, \alpha \in M \tag{1}$$

$$x_{j\sigma_t^j} \geq x_{j\sigma_{t-1}^j} + p_{j\sigma_{t-1}^j} \text{ for all } j \in J \text{ and } t = 2, \cdots, m \tag{2}$$

$$x_{i\alpha} \geq x_{j\alpha} + p_{j\alpha} \text{ or } x_{j\alpha} \geq x_{i\alpha} + p_{i\alpha} \text{ for all } i, j \in J, \alpha \in M. \tag{3}$$

To incorporate the objective function, we introduce an additional variable z that is constrained as follows:

$$z \geq x_{j\sigma_m^j} + p_{j\sigma_m^j} \text{ for all } j \in J. \tag{4}$$

The job-shop problem is then

$$\text{minimize } \{z: \text{ subject to } (1), (2), (3), \text{ and } (4)\}. \tag{5}$$

This formulation is a *disjunctive programming problem* (Balas[4]), due to the "either-or" constraints (3). By a straightforward transformation, we can turn this into a mixed-integer programming problem, introducing a binary variable Y_{ij}^{α} for each of the conditions (3) and imposing the new constraints

$$x_{i\alpha} \geq x_{j\alpha} + p_{j\alpha} - K \cdot Y_{ij}^{\alpha}$$
$$x_{j\alpha} \geq x_{i\alpha} + p_{i\alpha} - K \cdot (1 - Y_{ij}^{\alpha}) \text{ for all } i, j \in J, \alpha \in M \tag{6}$$

where K is some large constant. The interpretation is that Y_{ij}^{α} is 1 if i is scheduled before j on machine α, and 0 if j is scheduled before i.

We have developed cutting-plane procedures for both the disjunctive and mixed-integer formulations. Notice that in the former case, we must check, after solving a linear programming relaxation, whether or not the constraints (3) are satisfied. This differs somewhat from the normal cutting-plane approach (as in the mixed-integer formulation), where we check if the integer variables do indeed take on integer values.

In the disjunctive formulation, we begin with the linear programming problem

$$\text{minimize } \{z: \text{ subject to } (1), (2), \text{ and } (4)\}. \tag{7}$$

We then successively add inequalities to the formulation which are valid for all vectors x that represent schedules, but are not satisfied by the current optimal solution to the linear programming relaxation. The optimal value of the linear programming problems provide increasingly stronger lower bounds on the value of the optimal schedule. (For an account of this type of cutting-plane algorithm, we refer the reader to the text of Nemhauser and Wolsey[20].)

Similarly, in the mixed-integer programming formulation, writing the bounds on the Y's as

$$0 \leq Y_{ij}^{\alpha} \leq 1 \text{ for all } i, j \in J, \alpha \in M \tag{8}$$

we begin with the relaxation

$$\text{minimize } \{z: \text{ subject to } (1), (2), (4), (6), \text{ and } (8)\}. \tag{9}$$

In this case, we can make use of valid inequalities involving the Y variables as well as the inequalities developed for the disjunctive programming formulation.

The inequalities we considered are the following:

1. Basic cuts (Dyer and Wolsey,[11] McGee[18])

For each job j and machine α, let $E_{j\alpha}$ denote the earliest possible starting time of j on α (which is just the sum of j's processing times on the machines before α in j's prescribed ordering) and let $F_{j\alpha}$ denote the minimum completion time of j after it is processed on α (which is just the sum of j's processing times on the remaining machines). Not let $S \subseteq J$ be a subset of jobs and let $\alpha \in M$ be a machine, and let $E_{S\alpha}$ be the minimum of $E_{j\alpha}$ over all $j \in S$, let $F_{S\alpha}$ be the minimum of $F_{j\alpha}$ over all $j \in S$, and let $p_{S\alpha}$ be the sum of $p_{j\alpha}$ over all $j \in S$. Then by considering the order on which the jobs in S are scheduled on α, it is straightforward to check that the inequality

$$\sum_{j \in S} p_{j\alpha} x_{j\alpha} \geq E_{S\alpha} p_{S\alpha} + \sum_{i \neq j \in S} p_{i\alpha} p_{j\alpha} \tag{10}$$

is satisfied by all schedules. Similarly, we can obtain the "reverse" inequality

$$\sum_{j \in S} p_{j\alpha}(z - x_{j\alpha}) \geq F_{S\alpha} p_{S\alpha} + \sum_{j \in S} p_{j\alpha}^2 + \sum_{i \neq j \in S} p_{i\alpha} p_{j\alpha} \tag{11}$$

by considering the reverse of the order on which the jobs are scheduled. A similar "reverse" construction can be applied to all of the classes of inequalities given below (with the exception of the triangle cuts).

2. Two-job cuts (Balas[5])

Let i and j be distinct jobs and let α be a machine, and suppose that $E_{j\alpha} < E_{i\alpha} + p_{i\alpha}$ and $E_{i\alpha} < E_{j\alpha} + p_{j\alpha}$. Then the basic cut for the pair $\{i, j\}$ can be sharpened as follows:

$$(p_{i\alpha} + E_{i\alpha} - E_{j\alpha}) x_{i\alpha} + (p_{j\alpha} + E_{j\alpha} - E_{i\alpha}) x_{j\alpha} \geq p_{i\alpha} p_{j\alpha} + E_{i\alpha} p_{j\alpha} + E_{j\alpha} p_{i\alpha}. \tag{12}$$

To see that this is valid, it suffices to check that the earliest possible times for scheduling i and j on α satisfy the inequality (since, by assumption, the coefficients in front of the variables $x_{i\alpha}$ and $x_{j\alpha}$ are nonnegative and

any valid schedule must make $x_{i\alpha}$ and $x_{j\alpha}$ take on at least their minimum values). These earliest possible times are to set $x_{i\alpha} = E_{i\alpha}$ and $x_{j\alpha} = E_{i\alpha} + p_{i\alpha}$ (corresponding to scheduling i before j on α), or to set $x_{j\alpha} = E_{j\alpha}$ and $x_{i\alpha} = E_{j\alpha} + p_{j\alpha}$ (corresponding to scheduling j before i on α). By substituting these values into (12) we see that they do indeed satisfy the inequality.

3. Clique cuts (Balas[5])

Let $S \subseteq J$ and let $\alpha \in M$. As in the example above (where $|S| = 2$) we can write a family of valid inequalities, involving only the variables $(x_{j\alpha}: j \in S)$, which sharpen the basic cut for the pair S, α. To carry this out, let us form a matrix K having columns indexed by S and with a row corresponding to each permutation of S. The entries in each row are the earliest possible starting times of the jobs S on α, assuming that the jobs are scheduled on α in the order given by the permutation of S corresponding to this particular row. It is clear that if we look at any possible schedule $\bar{x}$, then there is a row of K such that the entries in the row are each less than or equal to the corresponding value $\bar{x}_{j\alpha}$. So any $\geqslant$ inequality, involving only the variables $(x_{j\alpha}: j \in S)$ and having only nonnegative coefficients, that is valid for each row of K is also valid for any vector representing a schedule. Thus, any vector in the polyhedron $P(S, \alpha) = \{t: Kt \geqslant 1, t \geqslant 0\}$, where 1 denotes the vector of all 1's gives a valid inequality for the set of all schedules. (We have chosen the right-hand-side of 1's, since we may scale any valid inequality with nonnegative coefficients to force the right-hand-side to be 1.) The strongest inequalities of this type correspond to the vertices of the polyhedron $P(S, \alpha)$, and we call this finite set the clique cuts for S and α. (For more details, see Balas.[5])

4. Two-job, two-machine cuts

Using the same approach as in the two-job cuts and the clique cuts, we can write valid inequalities involving two fixed jobs and two fixed machines. Although this approach can, in principle, be extended to larger subsets of jobs and machines, the size of the matrix involved quickly becomes too large to handle effectively.

5. Triangle cuts

For any three jobs i, j, and k, and any machine α, we have the simple triangle inequality

$$Y_{ij}^{\alpha} + Y_{jk}^{\alpha} + Y_{ki}^{\alpha} \leqslant 2 \tag{13}$$

arising from the fact that if i is scheduled before j and j is scheduled before k, then i must be scheduled before k. In writing this inequality, we may need to substitute, say, $1 - Y_{ji}^{\alpha}$ for Y_{ij}^{α}, depending on how we have assigned the Y variables, since we only create one of the two possible variables Y_{ij}^{α} and Y_{ji}^{α}. (This comment applies to all of the remaining classes of inequalities.)

6. Basic cuts plus epsilon

With the addition of the Y variables, we can use the following technique to tighten the basic cuts. Let $S \subseteq J$, $\alpha \in M$, and $k \in J$. (The job k may be in S.) Then, letting q^+ denote the maximum of q and 0 for any real number q, the inequality

$$\sum_{j \in S} p_{j\alpha} x_{j\alpha} \geqslant \sum_{i \neq j \in S} p_{i\alpha} p_{j\alpha} + E_{k\alpha} p_{S\alpha} - \left(\sum_{j \in S} Y_{jk}^{\alpha} \left(E_{k\alpha} - E_{j\alpha} \right)^+ \right) p_{S\alpha} \tag{14}$$

is valid for all schedules, since either k is scheduled on α before all (other) jobs in S, or it is not (in which case a penalty is paid on the right-hand-side of (14).

7. Half cuts

Again, let $S \subseteq J$ and let $\alpha \in M$. For each job $k \in S$ the inequality

$$x_{k\alpha} \geqslant E_{S\alpha} + \sum_{j \in S \setminus \{k\}} Y_{jk}^{\alpha} p_{j\alpha} \tag{15}$$

is valid for all schedules, since k cannot be processed on α until all jobs in S that are scheduled on α before k are completed. This class of inequalities is useful for pushing the values of the Y's away from $1/2$.

8. Late job cuts (Dyer and Wolsey[11])

Let $S \subseteq J$, $\alpha \in M$, $k \in S$, and $l \in J$. (The job l may be in S.) By considering the order in which k, l, and the remaining jobs in S are scheduled on α, we have that

$$x_{k\alpha} \geqslant E_{l\alpha} + \sum_{j \in S \setminus \{k\}} Y_{jk}^{\alpha} p_{j\alpha} - \sum_{j \in S} Y_{jl}^{\alpha} \left(E_{l\alpha} - E_{j\alpha} \right)^+ \tag{16}$$

is valid for all schedules. ■

These eight classes, and the corresponding "reverse" inequalities, are the only cutting-planes included in our study. (Several other classes are given in Dyer and Wolsey[11] and Queyranne.[21]) Our separation routines are straightforward, involving either complete enumeration or simple heuristics. The exception to this is our method for finding violated clique cuts, which requires the solution of linear programming subproblems as outlined above. For this class of inequalities, we limit our search to those sets $S \subseteq J$ and machines $\alpha \in M$ such that the corresponding basic cut is within some fixed tolerance of being violated.

We group the inequalities into three pools of cutting planes. The first, *Cuts* 1, consists of only the basic cuts.

The second, *Cuts* 2, consists of the basic cuts, the two-job cuts, the clique cuts (up to a maximum clique size of 5), and the two-job, two-machine cuts. These two pools require only the disjunctive formulation. The third pool, *Cuts* 3, consists of all of the above eight classes, and therfore requires the mixed-integer programming formulation. The values of the lower bounds we obtained using each pool are reported in Table I.

Each of the test problems has 10 machines and 10 jobs. The problem MT10 is the well-known 10 by 10 problem of Muth and Thompson; the problems ABZ5 and ABZ6 are two problems from Adams, Balas and Zawack[1]; the problems LA19 and LA20 are problems of Lawrence[15], reported in [1] as problems 19 and 20 of their Table 2; the problems ORB1 through ORB5 were generated in Bonn in 1986: the prescribed processing orders for each problem were created by guests (with the challenge to make the problems "difficult") and for each problem we generated two sets of processing times at random and retained that set which gave a problem having the greatest gap between the optimal value and a standard lower bound).

The entries in the table under the headings "Preempt" and "1-Mach" report the values of two standard lower bound techniques for the job-shop problem. The first is a *preemptive lower bound*, which consists of looking at each machine α individually as processing a set of jobs having release dates $E_{j\alpha}$ and finishing times $F_{j\alpha}$, with objective to minimize the maximum of the completion times, where we are allowed to interrupt a job while it is being processed and complete its processing at a later time. (For a discussion of this bound, see Carlier.[9]) The second, a *one-machine lower bound*, is defined in a similar manner, except that we are not permitted to interrupt a job while it is being processed. (Our implementation of this bounding procedure is based on the algorithm of Carlier.[9]) This bound is always at least as strong as the preemptive lower bound, and in some instances (although not in the test set reported in Table I) it is significantly stronger. In each case, we take the maximum of the bounds over all machines α. (For a detailed discussion of these techniques, see Lenstra[16] or Rinnooy Kan.[22])

The running times, on an IBM 3081D computer, are given in seconds, following each entry in the table. The linear programming problems were solved using the IBM package MPSX. In each of the three cut pools, we used a heuristic method for guiding our search for violated basic cuts. In Cuts 2 and Cuts 3, the clique cuts were generated by solving, for each subset S and machine α for which the basic cut was close to violation, a linear programming problem over the polyhedron $P(S, \alpha)$. (See Balas.[5])

In 8 out of the 10 problems, the cutting-plane bounds were superior to the standard methods, but in all cases they required a greater computational effort. For all problems, the observed improvement by moving from Cuts 1 to the larger cut pools was quite small, particularly when one considers the large jump in running time incurred from moving from one pool to the next. The computational difficulty is partly due to the fact that the clique cuts tend to have coefficients of large magnitude (which make the linear programming problems difficult to solve). One promising aspect of the cutting plane work is that in 6 of the 10 cases, the (quickly computable) bound of Cuts 1 is by itself superior to the standard bounding techniques.

It is clear that the running times of the cutting-plane implementations can be brought down considerably (using a more modern linear programming package), but it remains as a research challenge to find classes of valid inequalities that will close the large gap between the lower bound values and the optimal values of the scheduling problems, within a reasonable amount of computation time.

Table I
Ten 10 × 10 Problems: Lower Bounds

Problem	Preempt	1-Mach	Cuts 1	Cuts 2	Cuts 3	Optimal
MT10	808 (0.10)	808 (0.12)	823 (5.23)	824 (305.58)	827 (7552.34)	930
ABZ5	1029 (0.10)	1029 (0.12)	1074 (5.61)	1076 (611.29)	1077 (4971.42)	1234
ABZ6	835 (0.10)	835 (0.12)	835 (4.87)	837 (334.72)	840 (5257.35)	943
LA19	709 (0.10)	709 (0.11)	709 (5.57)	716 (917.07)	—	842
LA20	807 (0.10)	807 (0.12)	807 (5.13)	807 (806.16)	—	902
ORB1	929 (0.10)	929 (0.10)	930 (7.16)	931 (358.08)	—	1059
ORB2	766 (0.10)	766 (0.12)	768 (10.02)	769 (326.86)	—	888
ORB3	865 (0.10)	865 (0.11)	869 (5.95)	870 (449.00)	—	1005
ORB4	833 (0.10)	833 (0.12)	891 (5.58)	895 (555.51)	—	1005
ORB5	801 (0.10)	801 (0.12)	801 (6.90)	801 (323.23)	—	887

2. Branch and Bound

Branching schemes for the job-shop problem have been well-studied in the literature. One of the most successful schemes discussed can be derived directly from the disjunctive formulation: We choose some machine $\alpha \in M$ and a pair of jobs $i, j \in J$, and enforce the constraint (3), creating one subproblem, P_1, where i is scheduled before j on α and a second subproplem, P_2, where j is scheduled before i. The idea is that if we can make a good choice of α and i, j, then we settle some essential conflict in the schedule and thus make a significant improvement in the lower bound.

To test this scheme, we implemented the following simple algorithm: Suppose we have created the subproblems $P_1, \ldots, P_k$. Choose the (unprocessed) subproblem P_l such that preemptive lower bound, $LB(P_l)$, is minimum. Process P_l, creating the new subproblems P_{k+1} and P_{k+2}, by branching on the machine α and jobs i, j such that minimum $\{LB(P_{k+1}), LB(P_{k+2})\}$ is maximized. That is, naively try all possible choices of branching, and greedily choose that branch which gives the greatest direct increase in the lower bound. Whenever a problem P_l has $LB(P_l) \geqslant$ UB (the value of the best schedule known), it can be deleted from the list of subproblems. Surprisingly, given the history of the Muth and Thompson problem, this greedy procedure proved the (optimal) 930 lower bound for MT10 in just over 1 hour on a Sun Sparcstation 1. Based on this, and some testing of a preliminary implementation of the main competing branching scheme (based on "active schedule generation", see Lenstra[16] or Rinnooy Kann[22]), we decided to adopt the disjunctive formulation in our branch and bound work.

We then developed a more sophisticated branch and bound method using the following *edge-finding* algorithm based on the work of Carlier and Pinson[10]: At each node, if there exists a machine α, a subset $S \subseteq J$, and a job $i \in J$ such that

$$\min_{j \in S \setminus \{i\}} E_{j\alpha} + p_{S\alpha} + F_{S\alpha} \geqslant UB \qquad (17)$$

then i must be processed on α before any other job in S. The "reverse" inequality also applies. In addition, for each machine α, maintain the set $C_\alpha \subseteq J$ of the jobs not yet ordered for α, and subset $\hat{E}_\alpha$ (and $\hat{F}_\alpha$) $\subseteq C_\alpha$ of jobs in C_α that could be scheduled first (respectively last). Initially, $C_\alpha = \hat{E}_\alpha = \hat{F}_\alpha = J$. If there exists a machine α and a job $i \in \hat{E}_\alpha$ such that

$$E_{i\alpha} + p_{C_\alpha\alpha} + F_{\hat{F}_\alpha\alpha} \geqslant UB \qquad (18)$$

then i can be removed from $\hat{E}_\alpha$. If $\hat{E}_\alpha$ contains only one element i, then i must be processed on α before any other job in C_α.

Whenever (17) or (18), or their "reverse", imply that i must be processed before j on α, this eliminates one of the disjuncts in (3) from consideration, simplifying the problem. When all the disjuncts involving i on α have been decided, then remove i from C_α, since i is completely ordered for α.

When no more disjuncts can be decided by (17) or (18), then select a machine α and jobs i and j on which to branch. Restrict attention to the machine that had the worst initial preemptive bound until that machine is completely scheduled, and then consider all remaining machines. Subject to this, choose the set $\hat{E}_\alpha$ of $\hat{F}_\alpha$ of smallest cardinality, and within that set, choose a pair of jobs by computing:

$$d_{ij} = \max(0, E_{i\alpha} + p_{i\alpha} + p_{j\alpha} + F_{j\alpha} - LB) \qquad (19)$$

$$d_{ji} = \max(0, E_{j\alpha} + p_{j\alpha} + p_{i\alpha} + F_{i\alpha} - LB) \qquad (20)$$

$$a_{ij} = \min(d_{ij}, d_{ji}) \qquad (21)$$

$$v_{ij} = |d_{ij} - d_{ji}| \qquad (22)$$

and selecting the pair with maximum v_{ij}, breaking ties by selecting the pair with maximum a_{ij}.

The main difference between *edge-finder* and the algorithm described by Carlier and Pinson in [10] is that *edge-finder* tests (17) for all $S \subseteq J$, $i \in S$, and $\alpha \in M$, while Carlier and Pinson only test it for all doubletons S, and one additional heuristically determined S and $i \in S$ for each $\alpha \in M$. However, from comparisons of our computational experience with theirs, it appears that there are also many implementation differences.

3. Heuristics

In an optimization procedure, it is important to have a good heuristic method for obtaining an initial schedule. Thus, based on the computational experience reported in Adams, Balas and Zawack,[1] we implemented their "shifting-bottleneck" scheduling algorithm. A rough outline of the method is the following: Suppose we have scheduled all jobs J on the machines $\alpha_1, \ldots, \alpha_k$ (that is, for each of these machines, we have fixed the order on which we process the jobs). Now, for each of the remaining (unscheduled) machines, we calculate the values of $E_{j\alpha}$ and $F_{j\alpha}$, respecting the schedules on the machines $\alpha_1, \cdots, \alpha_k$, and compute its one-machine lower bound. We then fix the schedule of that machine which has the greatest lower bound, ordering the jobs as prescribed in the one-machine schedule, and perform a local reoptimization of the k scheduled machines (see [1]). The entire process is repeated until all m machines are scheduled.

This basic algorithm performs quite well, as indicated in Table II, under the heading "Bottle." (The running times, in seconds on a Sun Sparcstation 1, are reported after the values of the schedules.) Moreover, Adams, Balas and Zawack [1] were able to produce still better solutions by adding a certain backtracking scheme and

Table II
Ten 10 × 10 Problems: Heuristics

Problem	Bottle	Bottle-4	Bottle-5	Bottle-6	Shuffle
MT10	952 (2.7)	952 (11.4)	944 (46.0)	944 (218.6)	930* (57.6)
ABZ5	1270 (2.3)	1250 (27.4)	1245 (100.3)	1245 (503.5)	1245 (113.7)
ABZ6	952 (2.8)	952 (22.0)	943* (67.3)	943* (351.3)	943* (68.0)
LA19	863 (2.9)	863 (35.1)	848 (125.7)	847 (657.5)	848 (161.6)
LA20	918 (2.9)	918 (16.9)	918 (97.6)	913 (491.2)	911 (133.2)
ORB1	1176 (2.6)	1125 (25.2)	1092 (87.8)	1073 (265.9)	1070 (103.1)
ORB2	927 (3.0)	894 (31.7)	894 (117.6)	894 (497.0)	890 (194.1)
ORB3	1090 (2.7)	1053 (16.4)	1031 (59.1)	1022 (262.1)	1021 (137.8)
ORB4	1065 (2.6)	1040 (26.6)	1031 (63.1)	1019 (236.5)	1019 (78.1)
ORB5	939 (1.1)	899 (16.0)	896 (68.1)	896 (277.6)	896 (128.0)

allowing for more computation time. Thus, in our implementation, for each of the final t machines scheduled (where t is an input parameter), we try not only the machine having the greatest lower bound, but each of the other unscheduled machines as well. (That is, for all $s = 1, \ldots, t$, after scheduling $m - s$ machines, we try each of the remaining s machines as the $m - s + 1$'st machine to be scheduled.) The performance of this method for $t = 4$, 5, and 6 is reported in Table II under "Bottle-4," "Bottle-5," and "Bottle-6." These values are slightly worse then those obtained by the more sophisticated method used in [1], but they reflect the possible improvements one can obtain by a repeated use of the basic algorithm.

Although the values reported in Table II and in [1] are very good, in most cases the best solutions obtained are still far from optimal. Thus, there may be some room for improvement, particularly when one considers that increasing the level of backtracking adds a considerable amount of computation time to the process. With this in mind, we developed a method which takes a given schedule and attempts to produce one of shorter length.

The basic idea of our method, as suggested in Carlier and Pinson,[10] is that the kernel of the edge-finder algorithm can quite often quickly complete a full schedule, once the schedules of a relatively small number of machines are fixed. The algorithm, *shuffle*, we adopted proceeds as follows. Given a full schedule, we fix the processing orders of the jobs on a small number, t, of the machines and apply edge-finder to optimally complete the schedule. If the new schedule is shorter then the original, then we repeat the process, with the restriction that we do not choose to fix any of the machines whose schedules we have currently held fixed.

The best strategy for choosing the t machines to fix is not at all clear. In our initial implementation we randomly selected the machines, but this seldom led to an improved schedule. After some experimenting, we adopted a strategy based on the trivial lower bounds, $E_{j\alpha} + p_{j\alpha} + F_{j\alpha}$. To start off, for each machine we calculate the maximum of these bounds, where the E's and F's are computed with respect to holding the processing orders on all remaining machines fixed. In the first pass, we select those t available machines having the greatest bounds. After iterating with this selection rule until no further improvement in the schedule is obtained, we then make a second pass where we reverse the rule and choose the t machines having the minimum lower bounds.

Our rule for choosing the number, t, of machines to fix is dictated by the need to have enough structure to allow edge-finder to rapidly fill in the remainder of the schedule, while, at the same time, leaving a sufficient amount of freedom for improving the processing orders. From empirical observations, we choose $t = 1$ if $m = 10$, and $t = 2$, 3 or more if $m = 15$, and $t = 5$ or more if $m = 20$.

The values and running times of the heuristic are reported in Table II under the heading "Shuffle." In these runs, we gave the bottle-5 shedule as input to shuffle (thus the reported running times are the sum of the bottle-5 time and the shuffle time). In one instance, ORB3, the first pass of shuffle was halted after the maximum allowed computation time of 60 seconds, without completing a schedule. For each of the other problems, both passes terminated after only a modest amount of computation, with solutions comparable to the longer backtracking runs of bottle-6.

4. Results

Our optimization routine consists of a run of bottle-5, followed by a run of shuffle, to determine the initial bound, then a run of edge-finder to find and prove the optimal solution. The total running times for the ten 10 by 10 problems, on a Sun Sparcstation 1, are reported in Table III. In all cases, the problems were solved in under an hour of computation time. One point worth noting is that, judging by the times in Table III, it is apparent that

Table III
Solution of Ten 10 × 10 Problems

Problem	Heuristic	Optimal	Running Time (Seconds)	No. Nodes	Max Depth
MT10	930	930	372.4	16055	47
ABZ5	1245	1234	951.5	57848	68
ABZ6	943	943	90.9	1269	37
LA19	848	842	1462.3	93807	88
LA20	911	902	1402.3	81918	97
ORB1	1070	1059	1482.6	71812	58
ORB2	890	888	2484.6	153578	91
ORB3	1021	1005	2297.6	130181	88
ORB4	1019	1005	1013.3	44547	43
ORB5	896	887	526.0	23113	74

Table IV
Edge-Finder on MT10

Initial Bound	Running Time (Seconds)	No. Nodes	Max Depth
10000	5676.1	310392	339
1000	4951.5	268903	87
950	833.3	43778	71
940	559.5	28954	61
929	314.8	16055	47

the problem MT10 is not more difficult than many other 10 by 10 problems one can construct.

The quality of the solution found by shuffle is crucial in the optimization routine. This is demonstrated in Table IV, where we compare the times of edge-finder on problem MT10 with successively stronger upper bounds.

Including the ten 10 by 10 problems, we tested our routines on a total of 53 problem instances (3 from Muth and Thompson,[19] 5 from Adams, Balas and Zawack[1] (ABZ5-9), 40 from Lawrence[15] (LA1-40), and 5 that we generated ourselves (ORB1-5)). As reported in [1], the problems ABZ5-9 (following the numbering in Table 1 of [1]) and 21 of the problems LA1-40 (following the number of Table 2 of [1]), were unsolved.

We were able, through a combination of bottle, shuffle, and edge-finder, to solve all but 7 of these instances. In Table V, we report the best solutions and lower bounds we obtained for each of the 7 remaining open problems. In each case, the solution was obtained by a run of shuffle and the bound was obtained by edge-finder (except for LA27, which is the same one-machine lower bound as reported in [1]). We also included the optimal values of three problems from the test set that were particularly difficult for us to solve.

Each of the 10 problems in Table V poses a more

Table V
Ten Tough Job-Shop Problems

Problem	Size	Best Solution	Best Lower Bound	Status
ABZ7	15 × 20	668	654	OPEN
ABZ8	15 × 20	687	635	OPEN
ABZ9	15 × 20	707	656	OPEN
LA21	10 × 15	1053	1040	OPEN
LA24	10 × 15	935	935	SOLVED
LA25	10 × 15	977	977	SOLVED
LA27	10 × 20	1269	1235	OPEN
LA29	10 × 20	1195	1120	OPEN
LA38	15 × 15	1209	1184	OPEN
LA40	15 × 15	1222	1222	SOLVED

difficult computational challenge than the 10 by 10 problem MT10 (at least for out techniques). For example, to establish the 930 lower bound on MT10, edge-finder required 16,055 nodes in its search tree, whereas to establish the bound of 935 for LA24, it required 16,115,842 nodes, a difference of factor of 1000.

The routines bottle, shuffle, and edge-finder were all coded in the C programming language. The source codes to these routines, as well as the data sets for the 53 problem instances, are available, by request, from the authors. (The cutting-plane methods were coded in PL1 and depend heavily on the interface to IBM's MPSX package, and therefore are not really portable.)

ACKNOWLEDGMENTS

This research was sponsored in part by an ONR Fellowship and in part by the National Science Foundation under contract numbers CCR-8805199 and CCR-9007602 (to D.A.).

We would like to thank Tim McGee and Ed Sewell, who were involved in the initial computational work on cutting-plane

methods, and to thank Professor Egon Balas and Professor Jan Karel Lenstra for kindly providing us with data sets of job-shop test problems.

REFERENCES

1. J. ADAMS, E. BALAS and D. ZAWACK, 1983. The Shifting Bottleneck Procedure for Job Shop Scheduling, *Management Science 34*, 391–401.
2. S. ASHOUR and S.R. HIREMATH, 1973. A Branch-and-Bound Approach to the Job-Shop Scheduling Problem, *International Journal of Production Research 11*, 47–58.
3. E. BALAS, 1969. Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm, *Operations Research 17*, 941–957.
4. E. BALAS, 1979. Disjunctive Programming, *Annals of Discrete Mathematics 5*, 3–51.
5. E. BALAS, 1985. On the Facial Structure of Scheduling Polyhedra, *Mathematical Programming Study 24*, 179–218.
6. J.R. BARKER, 1981. Primal Search Tree Algorithms for the General Job Shop Problem, Ph.D. thesis, University of New South Wales, Kensington.
7. J.R. BARKER and G.B. MCMAHON, 1985. Scheduling the General Job-Shop, *Management Science 31*, 594–598.
8. P. BRATLEY, M. FLORIAN and P. ROBILLARD, 1970. On Sequencing with Earliest Starts and Due Dates with Application to Computing bounds for $(n/m/G/F_{\max})$ Problem, *Naval Research Logistics Quarterly 20*, 57–67.
9. J. CARLIER, 1982. The One-Machine Sequencing Problem, *European Journal of Operational Research 11*, 42–47.
10. J. CARLIER and E. PINSON, 1989. An Algorithm for Solving the Job-Shop Problem, *Management Science 35*, 164–176.
11. M. DYER and L.A. WOLSEY, 1990. Formulating the Single Machine Sequencing Problem with Release Dates as a Mixed Integer Program, *Discrete Applied Mathematics 26*, 255–270.
12. M. L. FISHER, B.J. LAGEWEG, J.K. LENSTRA and A.H.G. RINNOOY KAN, 1983. Surrogate Duality Relaxation for Job Shop Scheduling, *Discrete Applied Mathematics 5*, 65–67.
13. M. FLORIAN, P. TRÉPANT and G. MCMAHON, 1971. An Implicit Enumeration Algorithm for the Machine Sequencing Problem, *Management Science 17*, B782–B792.
14. B.J. LAGEWEG, J.K. LENSTRA and A.H.G. RINNOOY KAN, 1977. Job-Shop Scheduling by Implicit Enumeration, *Management Science 24*, 441–450.
15. S. LAWRENCE, 1984. Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques, GSIA, Carnegie Mellon University.
16. J.K. LENSTRA, 1976. *Sequencing by Enumerative Methods*, Mathematical Centre Tract 69, Mathematisch Centrum, Amsterdam.
17. J.K. LENSTRA and A.H.G. RINNOOY KAN, 1979. Computational Complexity of Discrete Optimization Problems, *Annals of Discrete Mathematics 4*, 121–140.
18. T. MCGEE, 1986. Personal communication.
19. J.F. MUTH and G.L. THOMPSON (eds.) 1963. *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, NJ. 1963.
20. G.L. NEMHAUSER and L.A. WOLSEY, 1988. *Integer and Combinatorial Optimization*, Wiley, New York.
21. M. QUEYRANNE, 1988. Structure of a Simple Scheduling Polyhedron, Working Paper No. 1277, Faculty of Commerce and Business Administration, The University of British Columbia.
22. A.H.G. RINNOOY KAN, 1976. *Machine Scheduling Problems: Classification, Complexity and Computations*, Nijhoff, The Hague.