Bilevel Programming Applied to the Flow Shop Scheduling Problem

John K. Karlof
Mathematical Sciences Department
The University of North Carolina at Wilmington

and

Wei Wang
Cascade Fibers Company
Sanford, North Carolina

SCOPE AND PURPOSE

Bilevel programming is a model for two independent levels of hierarchical decision making. Each decision maker attempts to optimize its own objective function and is affected by the actions of the other planner. There is a top planner and a bottom planer. The lower planner makes its decisions after, and in view of, the decisions of the top planner. The standard flow-shop scheduling problem is to schedule n jobs on m machines where each job is processed on each machine and each job follows the same ordering of machines. The objective function is usually to minimize the makespan (the total time from when the first job begins to when the last job finishes) or the total flowtime (the sum of the times it takes each job to be processed through the sequence of machines). We alter the flow-shop scheduling problem by adding operators and imposing a hierarchy of two decision makers. The top planner (the shop owner) assigns the operators to the machines in order to minimize her objective, the total flowtime, while the lower planner (the customer) decides on the jobs' schedule in order to minimize his objective, the makespan. We model the problem as a bilevel programming problem and develop a two level branch and bound algorithm for its solution.

ABSTRACT

A two level branch and bound algorithm is developed to solve an altered form of the standard flow-shop scheduling problem modeled as a bilevel programming problem. The flow-shop scheduling problem considered here differs from the standard problem in that operators are assigned to the machines and each operator has a different time table for the jobs on each machine. The shop owner is considered the top planner and assigns the operators to the machines in order to minimize the total flowtime while the customer is the bottom planner and decides on a job schedule in order to minimize the makespan.

1. INTRODUCTION

The standard mathematical programming problem involves finding an optimal solution for just one decision maker. But many planning problems contain an hierarchical decision structure, each with independent, and often conflicting objectives. These types of problems can be modeled using a multilevel programming approach. Bilevel programming is the simpliest class of multilevel programming problems in which there are two independent decision makers. Recently, several different algorithms have been proposed to solve the bilevel linear programming problem [1, 2]. Many of them are variations of the simplex method, although difficulties arise because the problem is not convex.

The standard flow-shop scheduling problem is to schedule n jobs on m machines where each job is processed on each machine and each job follows the same ordering of machines. The objective function is usually to minimize the makespan, the total flowtime, or some combination of these.

The goal of this paper is to apply the bilevel progamming idea to the standard flow-shop scheduling problem except that we alter the problem by adding operators and imposing a hierarchy of two decision makers. Suppose we have m machine operators in the shop and each of them has his own time table for different jobs on each machine. We assume the shop owner assigns the operators to the machines in order to minimize her objective, the total flowtime. After knowing the operator time schedule which is posted by the shop owner, the customer, who has the n jobs to be processed, decides the jobs' ordering to minimize his objective, the makespan. We develop a two level branch and bound algorithm to solve this bilevel mathematical programming problem.

Section 2 contains an introduction to bilevel programming. Section 3 contains a description of the standard flow shop scheduling problem, our altered version and the bilevel programming model. In section 4, we develop a branch and bound algorithm to solve the new version and section 5 contains computational results.

2. BILEVEL PROGRAMMING

Bilevel programming problems are characterized by two levels of hierarchical decision making. The top planner makes its decision in full view of the bottom planner. Each planner attempts to optimize its objective function and is affected by the actions of the other planner. The properties of bilevel programming problems are summarized as follows: [2]

1. The system has interacting decision making units within a hierarchical structure.

2. The lower unit executes its policies after, and in view of, the decisions of the higher unit.

3. Each level maximizes net benefits independently, no compromise is possible.

4. The effect of the upper decision maker on the lower problem is reflected in both its objective function and set of feasible decisions.

Let x_1 be a vector variable indicating the higher decision level's choice and x_2 be a vector variable indicating the lower decision level's choice. Let S be the set of feasible choices $\{(x_1, x_2)\}$. For any fixed choice of x_1, level two will choose a value of x_2 that optimizes the level two objective function $f_2(x_1, x_2)$. Hence, for each feasible value of x_1, level two will react with a corresponding value of x_2. This results in a functional relationship between the decisions of level one and the reactions of level two, say $x_2 = \Psi(x_1)$.

So the bilevel programming problem may be formulated as:

$$\begin{array}{ll} \max_{x_1} f_1(x_1, x_2) & \text{where } x_2 \text{ solves} \\ \max_{x_2} f_2(x_1, x_2) & \text{st: } (x_1, x_2) \in S \end{array}$$

Then $S_1 = \{(x_1, x_2^*) | f_2(x_1, x_2^*) = \max_{x_2} f_2(x_1, x_2)\}$ is the level one feasible region and S is the level two feasible region.

3. THE FLOW SHOP SCHEDULING PROBLEM

The standard flow shop problem is one of scheduling n jobs in a shop containing m machines, where each job contains m operations and every job follows the same ordering of machines as it is processed. A variety of performance measures for evaluating the quality of a derived sequence of jobs have been developed such as miminizing job idleness, minimizing the number of machine idle periods, and minimizing makespan, or total flow time [3, 4, 5, 6, 7]. In this paper, we consider makespan and flowtime. The makespan is the time from when the first job begins on the first machine until when the last job finishes on the last machine. The flowtime of each job is the time from when the first job begins on the first machine until the time when that job finishes on the last machine. The total flowtime is the sum of the flowtimes of the jobs.

We consider the following variation of the standard problem. Suppose there are m operators in the shop and each of them has his own time table for different jobs on each machine. We assume the shop owner has to pay the operators based on the total flowtime of the jobs, but the customer's charges are based on the makespan of the jobs. Thus the objective of the shop owner is to minimize the total flowtime while the objective of the customer is to minimize makespan. After seeing the posted operator time schedule, the customer has to decide the jobs' ordering. Thus, for each arrangement of the $m!$ operators' schedules, the customer has $n!$ ways to arrange the jobs' order. This situation is an hierarchical decision process. The top planner is the shop owner and the customer, reacting to the shop owner's decision, is the bottom planner. The problem is formulated as a bilevel programming problem:

Define:

$mk_{i,j}$ = makespan associated with operator's schedule i and job schedule j.

$fl_{i,j}$=flowtime associated with operator's schedule i and job schedule j.

$$k_i = \begin{cases} 1, \text{ if operator schedule i is chosen.} \\ 0, \text{ otherwise.} \end{cases} \qquad r_j = \begin{cases} 1, \text{ if job schedule j is chosen.} \\ 0, \text{ otherwise.} \end{cases}$$

$$Minimize: \quad \sum_{i,j} fl_{ij} \cdot r_j \cdot k_i$$

where r_j solves

$$Minimize: \quad \sum_{i,j} mk_{ij} \cdot r_j \cdot k_i$$

subject to

$$\sum_{j=1}^{n!} r_j = 1$$

$$\sum_{i=1}^{m!} k_i = 1$$

$$k_i \in \{0,1\}, \;\; i = 1, \ldots, m!$$

$$r_j \in \{0,1\}, \;\; j = 1, \ldots, n!$$

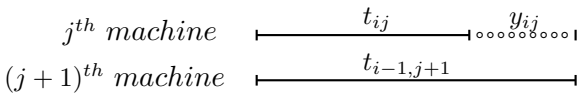
To understand this procedure better, we present a simple example which contains only three machines with three operators, and three jobs to be processed. In the following table, each row contains the optimal makespan and corresponding flowtime for the job orderings heading the column and for the operator schedule heading the row. The six feasible solutions for the shopowner are underlined, and the final optimal solution is boxed— *cba* with job order 213.

	123		132		213		231		312		321	
	mk	fl	mk	fl	mk	fl	mk	fl	mk	fl	mk	fl
abc	49	84	49	89	51	83	51	86	**46**	**81**	48	86
acb	48	83	50	91	51	86	53	89	**47**	**83**	49	86
bac	**46**	**93**	50	100	47	79	52	101	49	97	49	97
bca	47	82	47	88	50	82	48	82	**44**	**82**	47	90
cab	45	76	47	89	48	83	47	82	**43**	**82**	46	91
cba	45	81	49	89	**44**	**80**	49	88	47	85	46	86

4. THE BILEVEL PROGRAMMING ALGORITHM

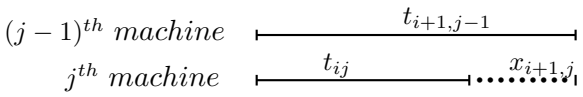
4.1 Definitions

Suppose an operators' schedule and a corresponding jobs' schedule have been assigned. Let t_{ij} be the processing time of the job in the i^{th} position on the j^{th} machine for this particular schedule. If a job in the i^{th} position finishes operation on the j^{th} machine, and the $(j+1)^{th}$ machine is still busy, then this job has to wait on the j^{th} machine until the time when the $(j+1)^{th}$ machine is free.



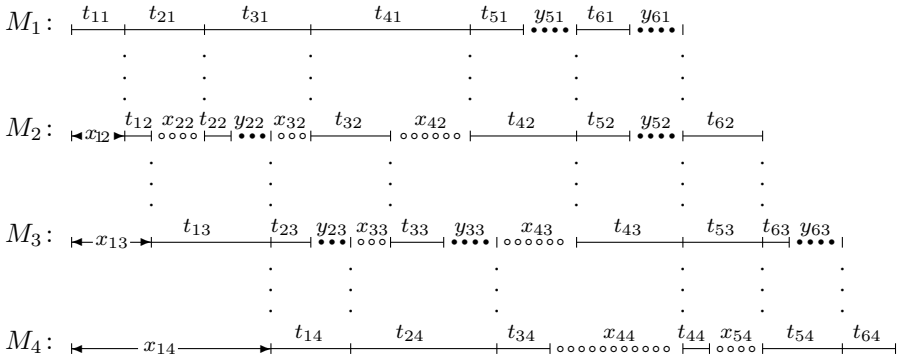
This time interval, denoted y_{ij}, is called *job idle time*. y_{ij} is the idle time for the job in the i^{th} position between the end of the operation on the j^{th} machine and its start on the $(j+1)^{th}$ machine. Because there is no job idle time on the last machine, y_{im}=0, for $1 \leq i \leq n$.

If the job in the i^{th} position finishes operation on the j^{th} machine and goes to the next machine or off the processing line if $j = m$, but the job which is in the $(i+1)^{th}$ position is still busy on the $(j-1)^{th}$ machine, then we have machine idle time.



The j^{th} machine is idle until the job which is in the $(i+1)^{th}$ position

Figure 4.1: Processing Schedules for Four Machines and Six Jobs



finishes operation on the $(j-1)^{th}$ machine. This time interval, denoted by $x_{i+1,j}$, is called *machine idle time*. $x_{i+1,j}$ is the idle time on the j^{th} machine between the end of the job in the i^{th} position on the j^{th} machine and the start of the job in $(i+1)^{th}$ position on the j^{th} machine. Because there is no machine idle time on the first machine, x_{i1}=0, for $1 \leq i \leq n$. We now present a diagram of the processing schedule for four machines and six jobs in order to illustrate x_{ij} and y_{ij}.

The makespan, denoted by mk, is the time from when the first job begins on the first machine until when the last job finishes on the last machine. From figure 4.1, we can see that:

$$mk = \sum_{i=1}^{n}(x_{im} + t_{im}) = \sum_{i=1}^{n} x_{im} + \sum_{i=1}^{n} t_{im}$$

We note that, independent of the jobs' ordering, $\sum_{i=1}^{n} t_{im}$ is fixed. So minimizing the makespan is equivalent to minimizing $\sum_{i=1}^{n} x_{im}$. The flowtime of

each job is the time from when the first job begins on the first machine until the time when that job finishes on the last machine. Then the total flowtime, denoted fl, is the sum of the flowtimes of each job. From figure 4.1, we can see that:

- the flowtime of the job in the first position is

$$fl_1 = x_{1m} + t_{1m};$$

- the flowtime of the job in the second position is

$$fl_2 = x_{1m} + t_{1m} + x_{2m} + t_{2m};$$
$$\vdots$$

- the flowtime of the job in the n^{th} position is

$$fl_n = x_{1m} + t_{1m} + x_{2m} + t_{2m} + \ldots + x_{nm} + t_{nm};$$

Therefore

$$\begin{aligned} fl &= fl_1 + fl_2 + \ldots + fl_n \\ &= n(x_{1m} + t_{1m}) + (n-1)(x_{2m} + t_{2m}) + \ldots + x_{nm} + t_{nm} \\ &= \textstyle\sum_{i=1}^{n}(n+1-i)(x_{im} + t_{im}) \end{aligned}$$

4.2 Algorithm 1: Calculating x_{ij} and y_{ij}

We incorporate the following observations in algorithm 1.

1. There is no machine idle time on the first machine.

   $x_{i1} = 0$ for $1 \leq i \leq n$

2. There is no job idle time on the m^{th} machine.

   $y_{im} = 0$ for $1 \leq i \leq n$

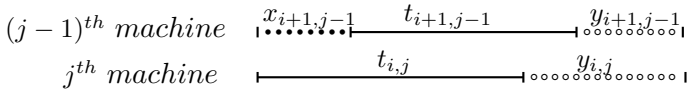
3. There is no job idle time for the job in the first position.

   $y_{1j} = 0$ for $1 \leq j \leq m$

4. $x_{1j} = \sum_{i=1}^{j-1} t_{1i}$ for $2 \leq j \leq m$

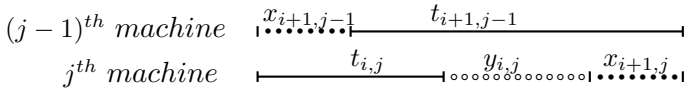
5. If $t_{ij} + y_{ij} \geq x_{i+1,j-1} + t_{i+1,j-1}$ then

   $y_{i+1,j-1} = (t_{ij} + y_{ij}) - (x_{i+1,j-1} + t_{i+1,j-1})$.



6. If $t_{ij} + y_{ij} < x_{i+1,j-1} + t_{i+1,j-1}$ then

   $x_{i+1,j} = (x_{i+1,j-1} + t_{i+1,j-1}) - (t_{ij} + y_{ij})$.



Algorithm 1:

Step 1: $x_{ij} = 0$, $y_{ij} = 0$ for $1 \leq i \leq n$, $1 \leq j \leq m$

Step 2: $x_{1j} = \sum_{k=1}^{j-1} t_{1k}$ for $2 \leq j \leq m$.

Step 3: Set $i = 1$, $j = 1$.

Step 4: If $t_{i,j+1} + y_{i,j+1} - t_{i+1,j} - x_{i+1,j} \geq 0$, then

$$y_{i+1,j} = t_{i,j+1} + y_{i,j+1} - t_{i+1,j} - x_{i+1,j} \ ;$$

otherwise

$$x_{i+1,j+1} = t_{i+1,j} + x_{i+1,j} - t_{i,j+1} - y_{i,j+1}.$$

Step 5: If $j < m$, set $j = j + 1$ and go to step 4.

Step 6: If $i < n$, set $i = i + 1$, $j = 1$ and go to step 4.

Step 7: Stop

4.3 Algorithm 2: Minimizing Makespan

For a given operators' schedule and n jobs to process there are $n!$ ways to arrange these jobs. Our purpose is to find an ordering that will result in the smallest makespan given the operators' schedule. We propose a branch and bound algorithm.

As we have already shown, the makespan,

$$mk = \sum_{i=1}^{n}(t_{im} + x_{im}).$$

Suppose the current best solution for this operators' schedule is mk_c and we are in the midst of forming a tree whose branches correspond to job orderings. From the last line of figure 4.1, it is evident that if the j^{th} job is in the first position and

$$x_{1m} + \sum_{i=1}^{n} t_{im} > mk_c,$$

then any makespan with the j^{th} job in the first position will be larger than mk_c regardless of how we arrange the rest of the jobs. Therefore we do not need to explore this branch any further. Otherwise, if

$$x_{1m} + \sum_{i=1}^{n} t_{im} \leq mk_c$$

then the next step for this branch is to compare $x_{1m} + x_{2m} + \sum_{i=1}^{n} t_{im}$ with mk_c. If

$$x_{1m} + x_{2m} + \sum_{i=1}^{m} t_{im} > mk_c$$

then again it is impossible to find a makespan which is smaller than mk_c on this branch. Then we would choose the next job to be processed in the second position. The worst situation is that all the jobs in the different positions need to be explored. Every time when we record a current best solution for the makespan we record the corresponding flowtime as well. This is because the two of them make a feasible solution.

The algorithm is shown below, note that we consider the operators' schedule as fixed. We use $\mathbf{k}$ to represent a particular job sequence and then $\mathbf{k} = \mathbf{k} + 1$ represents the next sequence. We use $mk_{\mathbf{k}}$ and $fl_{\mathbf{k}}$ to denote the makespan and the flowtime respectively for the current job sequence, mk_c and fl_c represent the current best makespan and flowtime respectively within this fixed operators' schedule, and we record the current best job ordering in $\mathbf{K}$.

Algorithm 2:

Initialization: Set $\mathbf{k}$ equal to the first job sequence. Calculate all x_{ij} and y_{ij} using algorithm 1 and calculate mk_k and fl_k. Set $mk_c = mk_{\mathbf{k}}$, $fl_c = fl_{\mathbf{k}}$, $\mathbf{K} = \mathbf{k}$, and $\mathbf{k} = \mathbf{k} + 1$.

Step 1: If $\mathbf{k}$ exceeds the last sequence, STOP.

Step 2: Do $i = 1, n$

Apply the recursive procedure in Algorithm 1 to find x_{im}.
Set $mk_i = \sum_{p=1}^{i} x_{pm} + \sum_{p=1}^{m} t_{pm}$
If $mk_i > mk_c$ then $\mathbf{k} = \mathbf{k} + 1$ (ie. prune off this branch) and goto step 1.
End do;
$mk_{\mathbf{k}} = mk_n$.

Step 3: Find $fl_{\mathbf{k}}$;

If $mk_{\mathbf{k}} < mk_c$ then

$mk_c = mk_{\mathbf{k}}$, $fl_c = fl_{\mathbf{k}}$,

$\mathbf{K} = \mathbf{k}$; $\mathbf{k} = \mathbf{k} + 1$; goto step 1.

If $mk_{\mathbf{k}} = mk_c$ and $fl_k < fl_c$ then

$mk_c = mk_{\mathbf{k}}$, $fl_c = fl_{\mathbf{k}}$,

$\mathbf{K} = \mathbf{k}$; $\mathbf{k} = \mathbf{k} + 1$; goto step 1.

4.4 Algorithm 3: Solution to the Bilevel Programming Problem

The set of feasible solutions for the upper level decision maker is a set of ordered pairs, one for each schedule of operators and each consisting of the best makespan for that operators' schedule together with the corresponding flowtime. We denote the current best ordered pair as $(\overline{mk}, \overline{fl})$. The final optimal solution is the ordered pair with the smallest flowtime.

During the following branch and bound procedure, we record the current smallest flowtime, $\overline{fl}$, from the current partial set of feasible solutions. If in the current branch representing an operators' schedule, we can not find a flowtime fl_k smaller than $\overline{fl}$, then no matter how small the makespan is on this branch, the optimal solution will not exist on this branch. Also, even if we find a flowtime on a branch smaller than $\overline{fl}$, it is not certain that this flowtime will replace $\overline{fl}$. This is because the smaller flowtime has to correspond to the smallest makespan on the branch; otherwise it is not a feasible solution for the upper level decision maker.

Recall $fl = \sum_{i=1}^{n}(n + 1 - i) \cdot (x_{im} + t_{im})$. Suppose jobs have been sched-

uled in positions $1, 2, \ldots, j$. We define $s(j, i)$ to be the $i^{\underline{th}}$ smallest of the remaining job processing times on the m^{th} machine and

$$\tilde{fl}_j = \sum_{i=1}^{j}(n+1-i)(x_{im}+t_{im}) + \sum_{i=j+1}^{n}(n+1-i)s(j, i-j).$$

If $\tilde{fl}_j > fl_c$, then regardless of how we schedule the remaining jobs we will not find a flowtime that is smaller than fl_c on this branch. Now if $\tilde{fl}_n < \overline{fl}$, then we found a flowtime smaller than our previous best feasible solution, but this flowtime may not correspond to the smallest makespan for this branch. So we need to begin at the beginning of the branch with the branch and bound method for makespan. Therefore, from this point, every time we start the branch and bound method for makespan, we are certain that there is a smaller flowtime on this branch. However, it is not guaranteed to be a feasible solution.

The branch and bound algorithm to solve the flowshop bilevel programming problem is given below. The notation $l = l + 1$ means go to the next sequence of operators' schedules. We record the current best operators schedule in L.

Algorithm 3:

Initialization: Set l = the first operators schedule. Find the best makespan for this operators schedule using algorithm 2 and calculate the corresponding flowtime. Set $(\overline{mk}, \overline{fl}) = (mk_c, fl_c)$.

Step 1: Set $l = l + 1$ and k = the first job sequence.

Step 2: Do $j = 1, n$

Apply the recursive procedure in Algorithm 1 to find x_{jm}.

Suppose k = the sequence $\{k_1, k_2, \ldots, k_n\}$.

If $\tilde{fl}_j > \overline{fl}$ then

prune off all branches beginning with $\{k_1, k_2, \ldots, k_j\}$ and set $k =$ to the next job sequence. If k exceeds the last sequence goto Step 1; otherwise repeat Step 2.

end if

end do

Step 3: If $\tilde{fl}_n < \overline{fl}$ then apply algorithm 2 to this operators schedule.

4.5 Example

We now present a simple example with three machines (and three operators) and four jobs. The time table of the three operators is as follows:

		job 1	job 2	job 3	job 4
operator 1	machine 1	1	6	7	2
	machine 2	5	1	8	6
	machine 3	9	10	6	5
operator 2	machine 1	9	8	10	7
	machine 2	3	9	4	7
	machine 3	1	3	6	9
operator 3	machine 1	5	8	5	8
	machine 2	6	8	6	2
	machine 3	5	6	8	2

Note: Comments are written in parentheses.

Algorithm 3:

Initialization: $l = 123$ (The first operator schedule.)

Algorithm 2:

Initialization: **k**=1234 (The first job schedule.) Calculate x_{ij} and y_{ij} using algorithm 2. (The results are summarized in figure 4.2) Then

$$mk_{\mathbf{k}} = \sum_{i=1}^{4}(t_{i3} + x_{i3}) = 32$$

and

$$fl_{\mathbf{k}} = \sum_{i=1}^{4}(4 + 1 - i) \cdot (x_{i3} + t_{i3}) = 93.$$

Set $mk_c = 32$ and $fl_c = 93$, **K**=1234 and **k**=1243.

Step 1: **k** does not exceed the last sequence.

Step 2: Consider $mk_i = \sum_{p=1}^{i} x_{p3} + \sum_{p=1}^{3} t_{p3}$. Apply algorithm 1 to find $x_{13} = 4$, then $mk_1 = 25$,$x_{23} = 7$, then $mk_2 = 32$, $x_{33} = 1$, then $mk_3 = 33 > mk_c$. This branch is pruned off. Step 2 is repeated for job schedules **k** = 1324, 1342, 1432. (They are all pruned off except for **k**=1432 since $mk_{1432} = 32$.)

Step 3: $fl_{1432} = 77$. So $mk_c = 32$, $fl_c = 77$, **K**=1432, and **k**=1423

Step 2: Step 2 is repeated for the remaining job sequences. (None of them are better than 1432. Algorithm 2 ends and we go to step 1 of algorithm 3.)

Set $(\overline{mk}, \overline{fl}) = (mk_c, fl_c) = (32, 77)$.

Step 1: Set $l = 132$, **k**=1234.

Figure 4.2: Gnatt chart

Step 2: Begin the recursive procedure in algorithm 1 and find $x_{13} = 7$

$$\begin{aligned} \tilde{fl}_1 &= 4(x_{13} + t_{13}) + \sum_{i=2}^{4}(4 + 1 - i)s(1, i - 1) \\ &= 4 \cdot 8 + 3 \cdot 3 + 2 \cdot 6 + 1 \cdot 9 \\ &= 63 \\ &< \overline{fl} \end{aligned}$$

Continue algorithm 1 and find $x_{23} = 7$

$$\begin{aligned} \tilde{fl}_2 &= \sum_{i=1}^{2}(4 + 1 - i)(x_{i3} + t_{i3}) + \sum_{i=3}^{4}(4 + 1 - i)s(2, i - 2) \\ &= 83 \\ &> \overline{fl} \end{aligned}$$

Therefore, prune all job branches that begin with job sequence 12. Set **k=** 1324 and repeat step 2. The values of x_{13} and $\tilde{fl}_1$ are the same as above, but $x_{23} = 6$ and $\tilde{fl}_2 = 83$. So prune off all branches beginning with the job sequence 13. Step 2 is repeated for job sequences 1432 and 1423 and these are pruned off as well. (Steps 1 and 2 (and if necessary 3) are repeated for the rest of the operator sequences. The current best solution turns out to be optimal.)

5. COMPUTATIONAL RESULTS

In this section, we present the results of computational experiments in which the algorithms in section 4 were used to solve several problems. The computations were done on a SUN Spark 10 and the code was written in C. The operator time tables were generated randomly choosing integers between one and ten. Thus the data for a problem with 5 machines and 8 jobs is five(one for each operator) 5 by 8 matrices in which the entries are integers between 1 and 10 chosen randomly. Representative ouput from the program is contained in the following table. The problem solved had 8 jobs and 5 machines. Each row of the table represents a current best solution. For this problem there are $5! \cdot 8! = 4,838,400$ possible branches in the tree. The algorithm only needed to calculate $230,558$ of them, roughly 5%. The total CPU time needed was 2 minutes and 44 seconds.

Makespan	Flowtime	Operator Assignment	Job Order
69	399	12345	54682731
70	384	12543	54832716
58	325	13245	53268741
59	322	15243	52743861

The next table contains total number of branches(TNB), number of branches calculated by the algorithm(BCA), the percent of the total branches that the algorithm calulated(PER), and CPU time for various sized test problems.

Number of Machines	Number of Jobs	TNB	BCA	PER	CPU (hours:min:sec)
5	5	14,400	2,652	18%	0:0:.4
5	7	604,800	19,827	3.3%	0:0:17
5	8	4,838,400	230,588	4.7%	0:2:44
7	7	25,401,600	135,466	.5%	0:9:42
5	10	435,456,000	10,265,783	2%	5:25:50
8	8	1,625,702,400	3,004,329	.18%	12:28:46

The largest problem which we ran the algorithm on was ten machines and ten jobs. It failed to come up with an answer after more than 19 hours of CPU time on the SUN and 2 hours of CPU time on a CRAY Y-MP. For problems this large an heuristic algorithm seems to be necessary.

References

[1] J. Bard and J. Moore, "A branch and bound algorithm for the bilevel programming problem," *SIAM Journal on Scientific and Statistical Computing*, **11**, 281-292, 1990.

[2] W. Bialas and M. Karwan, "Two-level linear programming," *Management Science*, **30**, 1004-1020, 1984.

[3] R. Ahmadi and U. Bagchi, "Minimizing job idleness in deadline constrained environments," *Operations Research*, **40**, 972-985, 1992.

[4] C. Liao, "Minimizing the number of machine idle intervals with minimum makespan in a flow-shop," *Journal of the Operational Research Society*, **44**, 817-824, 1993.

[5] R. Deckro, J. Hebert, and E. Winkofsky, "Multiple criteria job-shop scheduling," *Computers and Operations Research*, **9**, 279-285, 1982.

[6] W. Selen and D. Hott, "A mixed-integer goal-programming formulation of the standard flow-shop scheduling problem," *Journal of the Operational Research Society*, **37**, 1121-1128, 1986.

[7] J. Achugbue and F. Chin, "Scheduling the open shop to minimize mean flow time," *SIAM Journal on Computing*, **11**, 709-720, 1982.

[8] W. Wang, "Bilevel programming of the flow-shop scheduling problem," M.S. thesis, the University of North Carolina at Wilmington, 1993.