Operations Research

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems

Edward Ignall, Linus Schrage,

To cite this article:
Edward Ignall, Linus Schrage, (1965) Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems. Operations Research 13(3):400-412. http://dx.doi.org/10.1287/opre.13.3.400

APPLICATION OF THE BRANCH AND BOUND TECHNIQUE TO SOME FLOW-SHOP SCHEDULING PROBLEMS*

Edward Ignall and Linus Schrage

Cornell University, Ithaca, New York

(Received October 23, 1964)

The branch-and-bound technique of LITTLE, ET AL. and LAND AND DOIG is presented and then applied to two flow-shop scheduling problems. Computational results for up to 9 jobs are given for the 2-machine problem when the objective is minimizing the mean completion time. This problem was previously untreated. Results for up to 10 jobs, including comparisons with other techniques, are given for the 3-machine problem when the objective is minimizing the makespan.

AN M-machine job shop is said to be a flow shop if (1) all jobs are processed once and only once on each of the M machines and (2) a job's processing on machine A must precede its processing on machine B, its processing on machine B must precede its processing on machine C, etc. For $M=2$, the objective will be scheduling a given set of jobs so that the mean of their completion times is minimized. For $M=3$, the objective will be scheduling a given set of jobs so that the completion time of the last job to be completed (called makespan or schedule length) is minimized. Minimizing makespan is equivalent to minimizing idle time on the last machine. By use of the branch and bound technique (used by LITTLE, MURTY, SWEENEY, AND KAREL[1] on the traveling salesman problem and by LAND AND DOIG[2] on LP problems where some variables must have integer values), schedules that achieve these objectives can easily be obtained. It is felt the branch-and-bound technique is at least competitive with other methods (GIGLIO AND WAGNER,[3] DUDEK AND TEUTON[4]) for minimizing makespan. We know of no other method (short of complete enumeration) for the mean completion-time problem.

In order to use the branch-and-bound technique, one must be able to describe the problem as a tree, in which each node represents a partial solution. In addition, one must be able to write, at each node, a lower bound on the objective function (makespan or mean completion time) for all nodes that emanate from it. First it will be shown how the two flow-

* This work was supported by the National Science Foundation under Contract No. NSF-GP 2729.

shop problems that have been posed can be viewed as trees. Then a lower bound function for the 3-machine makespan problem will be given and the branch-and-bound technique described. A lower bound function for the 2-machine problem and computational results conclude the paper.

FLOWSHOP PROBLEMS AS TREES

IT HAS been shown (*see* JOHNSON[5]) for the 2-machine mean completion-time problem* and for the 3-machine makespan problem, that the jobs should be processed in the same order on all of the machines. Therefore, if there are n jobs in a jobset, each of the above problems is one of finding an optimum permutation or sequence of the integers $1, 2, \cdots, n$. The first node in the related tree structure corresponds to not having committed any of the n integers (i.e., jobs) to any position in the sequence. From this node there are n branches corresponding to the n possible integers (jobs) that can be assigned to the first position in the sequence. From each of these nodes there are $n-1$ branches corresponding to the $n-1$ integers available to be placed in the second position, etc. Thus one can see that there are $n!$ possible sequences or permutations, and $1+n+n(n-1)+\cdots+n!$ nodes in the entire tree.

A LOWER BOUND FOR THE MAKESPAN OF ALL NODES EMANATING FROM A GIVEN NODE

IN THE 3-machine makespan problem, each node represents a sequence of from 1 to n jobs. Consider node P, corresponding to *sequence* J_r, where J_r contains a particular subset (of size r) of the n jobs. Let TIME$A(J_r)$, TIME$B(J_r)$, and TIME$C(J_r)$ be the times at which machines A, B, and C respectively complete processing on the last of the r jobs in the sequence. Then a lower bound on the makespan of all schedules that begin with sequence J_r is:

$$LB(J_r)=\max\begin{bmatrix}\text{TIME}A(J_r)+\sum_{\bar{J}_r} a_i+\min_{\bar{J}_r}(b_i+c_i),\\ \text{TIME}B(J_r)+\sum_{\bar{J}_r} b_i+\min_{\bar{J}_r} c_i,\\ \text{TIME}C(J_r)+\sum_{\bar{J}_r} c_i\end{bmatrix},$$

where a_i, b_i, and c_i are the processing times of the ith job on machines A, B, and C respectively, and $\bar{J}_r$ is the *set* of $n-r$ jobs that have not been assigned a position in sequence J_r. $LB(P)=LB(J_r)$ is a lower bound on the makespan for any node that emanates from node P, since all such nodes represent sequences of from $r+1$ to n jobs that begin with sequence J_r.

* Although Johnson's 2-machine proof is for makespan, it clearly holds for mean completion time also.

THE BRANCH-AND-BOUND TECHNIQUE

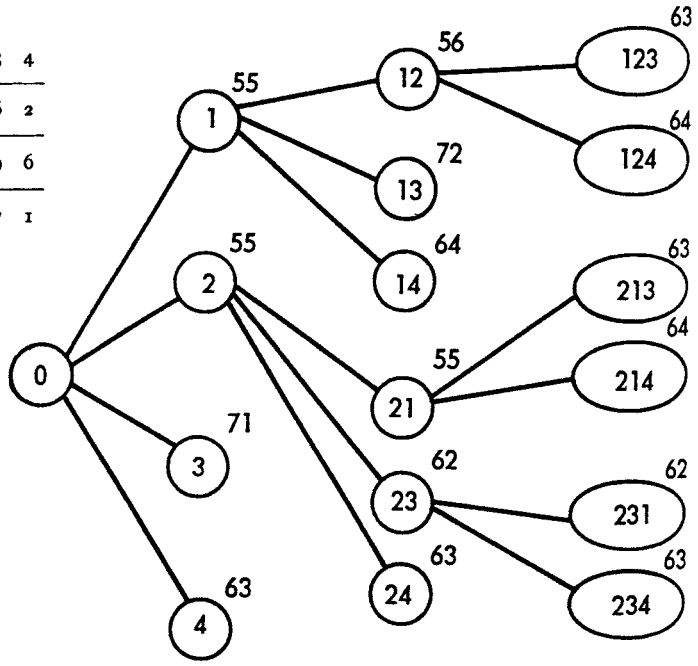
THE VERSION of the branch-and-bound technique that was used is basically this: Keep a list of nodes ranked by lower bound such that the node with the smallest lower bound is first. Also keep a set of attributes (the jobs that are in the sequence-in order, TIMEA, TIMEB, TIMEC, etc.) for each node. Begin by having on the list only the node that has scheduled none of the jobs. Update the list recursively as described below until an optimum is reached. (1) Remove the first node from the list. (2) Create a new node for every job that the 'just removed' node has not yet scheduled. Do this by attaching the unscheduled job to the end of the sequence of scheduled jobs. (3) Compute the lower bounds and other attributes for these newly created nodes and insert them ranked on the list. (4) Go to 1. The first time that a node that has scheduled all n jobs is first on the list, the problem is solved: that node's sequence is an optimal one.

An Example

Consider the following 4-job, 3-machine problem when the objective is minimizing makespan. The tree and list that are obtained in solving it by branch and bound are given below.

Jobset

i	1	2	3	4
a_i	13	7	26	2
b_i	3	12	9	6
c_i	12	16	7	1



Tree

LIST[a]

Node	Lower bound	TIMEB	TIMEC	Disposition
0	—	—	—	Branched from
1	55	16	28	Branched from
2	55	19	35	Branched from
3	71	35	42	
4	63	8	9	
21	55	23	43	Branched from
23	62	42	49	Branched from
24	63	25	36	
213	63	61	63	
214	64	57	64	
12	56	36	48	Branched from
13	72	48	55	
14	64	22	29	
123	63	61	63	
124	64	57	64	
231	62	60	62	An optimal sequence
234	63	51	63	

[a] The nodes are listed in the order of their creation; *not* in lower bound order.

In the example, if a newly created node and another node had the same lower bound, the newly created node was put *before* the old node on the list. The lower bounds are written next to the nodes on tree. Note that node 231's lower bound, which is the makespan for sequence 2314, is less than or equal to the lower bound of any other 'unbranched from' node in the tree (or on the list). For example, it would be impossible for a sequence beginning with job 4 to have makespan <63, and $62 \leqq 63$, so there is no need to explore any sequences that begin with job 4. The same applies to the other nodes, so 231 is optimal.

If either node 213 or node 214 (the two nodes emanating from node 21) had had a lower bound of 55, the problem would have been solvable with a minimum of effort. $1+4+3+2=10$ nodes would have been created and a maximum of $3+2+2=7$ nodes would have been on the list. In general, a minimum of $\frac{1}{2}n(n+1)$ nodes must be created and, in this case, $\frac{1}{2}n(n-1)+1$ nodes will be on the list. The worst possible case would require the creation of all $1+n+n(n-1)+\cdots+n!$ nodes while having $n!$ of them on the list.

DOMINATED NODES

THE AMOUNT of searching of the tree can be reduced still more—some nodes can be discarded even though the lower bound associated with these nodes may be less than the value of the optimal solution. In the 3-machine

makespan problem, let J_r and I_r be two different sequences, both containing the same r jobs. Clearly, TIME$A(J_r)$ = TIME$A(I_r)$. If TIME$B(J_r) \leqq$ TIME$B(I_r)$ and TIME$C(J_r) \leqq$ TIME$C(I_r)$, then any schedule which contains I_r at the beginning cannot be hurt by replacing I_r with J_r. We will say that J_r dominates I_r. The result is that node I_r can be discarded as soon as node J_r is created even if the lower bound for I_r is less than the value of the optimal solution obtained at the end of the calculations. Nodes can be discarded in much the same manner when the branch-and-bound technique is applied to the 2-machine mean completion time problem (and to the traveling salesman problem, etc.).

At the time it is discovered that a node is dominated, the work of creating it has already been done. The saving results from (1) not branching from dominated nodes and (2) not inserting them (or nodes that would result from branching from them) on the list. In our example, node 12 is dominated by node 21. Therefore, it was not necessary to branch from node 12 to nodes 123 and 124, even though $LB(12) = 56$, which is less than 62, the makespan of an optimal sequence. And nodes 12, 123, and 124 need not have been inserted on the list. In the 2-machine case, 10 jobsets of 8 jobs each* were run, first discarding dominated nodes (but not searching past I_r's position on the list or saving 'branched from' nodes—*see* next paragraph) and then without discarding them. When no dominance checks were made, the number of nodes created went up about 15 per cent and the maximum list size about 80 per cent. (In the first runs, about 30 per cent of all nodes created were discarded as dominated.) It was also observed that, the harder the jobset (in the sense that more nodes had to be created), the larger the percentage saving in nodes created and maximum list size resulting from discarding dominated nodes.

While searching down the list for the place to insert a newly created node, I_r, it is easy to check for dominance. If a node is discovered in this searching that dominates I_r, then I_r is discarded rather than inserted. (It should be clear that I_r cannot dominate a node with a smaller lower bound.) After I_r's place on the list has been found, the search down the list could be continued in an effort to find nodes that I_r dominates. Also, 'branched from' nodes could be saved on a separate list and compared with I_r to see if any of them dominates I_r. (Saving 'branched from' nodes would be necessary if node 12 were to be discarded in our example.) Both these last two procedures would involve more searching than is necessary just to

* For the manner in which these jobsets were generated, *see* the section "Generation of Jobsets for $M=2$, Mean Completion Time." The parameter values were $\rho=0$, $K=1$.

insert I_r in its proper slot on the list. Each of these two procedures (when added to searching down to I_r's place on the list) was compared to just searching down to I_r's place for the 10 jobsets mentioned in the previous paragraph. It was found that (1) searching past I_r's place on the list reduced the number of nodes created only 1 per cent while increasing computation time 30 per cent and (2) maintaining and searching a list of 'branched from' nodes reduced both the number of nodes created and the computation time by about 25 per cent. However, the two lists needed when 'branched from' nodes are saved required about 22 per cent more space than the list that resulted when those nodes were not saved.

On the basis of this analysis, searching past I_r's place on the list was eliminated as not worthwhile. In the 2-machine computations, checks for dominance were made only until I_r's place on the list was found. In the 3-machine computations, the saving and searching of 'branched from' nodes was added.

$M=2$; A LOWER BOUND ON MEAN COMPLETION TIME FOR ALL NODES EMANATING FROM A GIVEN NODE

Let a_i = processing time on machine A for job i,

b_i = processing time on machine B for job i,

d_i = completion time of job i (on machine B).

Let J_r denote a particular *permutation* of a set of r integers taken from $1, 2, \cdots, n$, and let $\bar{J}_r$ denote the *set* of integers that are in $1, 2, \cdots, n$ but not in J_r.

Consider any schedule that has J_r as its first r jobs. The sum of the completion times (which is equivalent to mean completion time) for this schedule can be divided into $\sum_{i\epsilon J_r} d_i + \sum_{i\epsilon J_r} d_i$. To obtain a lower bound for the second sum, define:

$$T_r \equiv \sum_{p=1}^{n-r} [\sum_{j\epsilon J_r} a_j + (n-r-p+1)a_{i_p} + b_{i_p}],$$

where $i_1, i_2, \cdots, i_{n-r}$ is a permutation of the integers in $\bar{J}_r$, and

$$S_r \equiv \sum_{p=1}^{n-r} [\max(d_k, \sum_{j\epsilon J_r} a_j + \min_{J_r} a_i) + (n-r-p+1)b_{j_p}],$$

where k is the last job in J_r and $j_1, j_2, \cdots, j_{n-r}$ is a permutation of the integers in $\bar{J}_r$. T_r is the sum of the completion times of the jobs in $\bar{J}_r$ when they are done in order $i_1, \cdots, i_{n-r}$ and if the times on machine A dominate. S_r is the same sum when the jobs are done in order $j_1, \cdots, j_{n-r}$ and if the times on machine B dominate. We discuss them below.

When $\min_{i\epsilon J_r} a_i \geqq \max_{i\epsilon J_r} b_i$ and $\min_{i\epsilon J_r} a_i \geqq d_k - \sum_{i\epsilon J_r} a_i$ and jobs are done in the order $i_1, i_2, \cdots, i_{n-r}$, then the completion time of job i_p is

$\sum_{j\in J_r} a_j + a_{i_1} + \cdots + a_{i_p} + b_{i_p}$ and the sum of the completion times for all the jobs in $\bar{J}_r$ is T_r. These conditions on the a_i, b_i, and d_k imply that the previous job is finished on machine B before the current one is finished on machine A. That is, each job starts on machine B immediately after being finished on machine A, with no delay being caused by the previous job on machine B. If the conditons are not met, then for any given permutation, delays are possible, and if they occur, the actual sum of completion times will exceed T_r. Therefore, the minimum possible value for T_r, call it $\hat{T}_r$, is a lower bound on $\sum_{J_r} d_i$ (no matter what the order in which the jobs in $\bar{J}_r$ are processed). Since $T_r = \text{constant} + \sum_p (n-r-p+1)a_{i_p}$, it is clear that choosing the permutation so that $a_{i_1} \leqq a_{i_2} \leqq \cdots \leqq a_{i_{n-r}}$ will give $\hat{T}_r$.

S_r is the sum of the completion times for the jobs in $\bar{J}_r$ when $\min_{J_r} b_i \geqq \max_{J_r} a_i$. This condition implies that a job's processing on machine B must be delayed by the preceding job's processing on machine B. $S_r = \text{constant} + \sum_{p=1}^{n-r} (n-r-p+1)b_{j_p}$ so it is clear that choosing a sequence so that $b_{j_1} \leqq b_{j_2} \leqq \cdots \leqq b_{j_{n-r}}$ will minimize S_r. Call the minimum value $\hat{S}_r$. In the same way as with T_r, it should be evident that if the condition does not hold, the actual sum of completion times will be $\geqq \hat{S}_r$, no matter how the jobs in $\bar{J}_r$ are sequenced.

Therefore, $LB(J_r) = \sum_{i\in J_r} d_i + \max(\hat{T}_r, \hat{S}_r)$ and $LB(\cdot)$ is a lower bound.

GENERATION OF JOBSETS FOR $M=2$, MEAN COMPLETION TIME

THE PROCESSING times were generated in the following way. Let $U_1, U_2, \cdots, U_n, V_1, V_2, \cdots, V_n$ denote a set of random numbers uniformly distributed on [0, 10]. a_i and b_i were computed as follows:

$$a_i = \text{the integer part of } (U_i) + G,$$

$$b_i = K \times \text{the integer part of } [(1-\rho)V_i + \rho U_i] + G.$$

K determines the relative work load on each machine, ρ the correlation between a job's first and second processing times, and G is a location parameter.

It might seem that G would have some effect on both the sequence obtained and the difficulty of obtaining it. However, it can be shown (*see* Appendix) that changing the value of G does not produce any change in either the optimal sequence or the path taken by the technique. So set $G=1$.

In fact, if the U_i and V_i were used directly, without rounding them down to integers, the length of the common interval on which U_i and V_i are distributed would also make no difference (*see* Appendix). Essentially, the optimal sequence and the difficulty of getting it depend only

on the relative locations of the a_i and b_i within the interval from $\min(\min_i a_i, \min_i b_i)$ to $\max(\max_i a_i, \max_i b_i)$.

As K increases from 1, the likelihood that $\min b_i \geqq \max a_i$ increases; as K decreases from 1, the likelihood that $\min a_i \geqq \max b_i$ increases. If either of these conditions on processing times exists, the scheduling problem has an easy analytical solution.

As ρ increases from 0, it becomes more likely that the jobs that have the long processing times on machine A will also have them on machine B. This implies these jobs belong at the end of the sequence. Similarly, short processing times on machine A will go with short ones on machine B, and

TABLE I

2 MACHINES, MEAN COMPLETION TIME—SUMMARY OF RESULTS

ρ	K	No. of jobs in jobset	No. of nodes created					Max no. of nodes on list				
			Min possible	Observed in 50 jobsets			Max possible	Min possible	Observed in 50 jobsets			Max possible
				Min[a]	Median	Max			Min	Median	Max	
0	1	5	15	15 (26)	15	38	206	11	11	11	21	120
0	1	6	21	21 (16)	29	108	1,237	16	16	20	56	720
0	1	7	28	28 (5)	61	190	8,660	22	22	43	98	5,040
0	1	8	36	36 (3)	123	777	69,281	29	29	80	293	40,320
0	1	9	45	45 (5)	321	1258	623,530	37	37	161	579	362,880
0.2	1	8	36	36 (8)	82	352	69,281	29	29	58	154	40,320
0.5	1	8	36	36 (15)	45	212	69,281	29	29	36	133	40,320
0	0.6	8	36	36 (47)	36	73	69,281	29	29	29	53	40,320
0	0.8	8	36	36 (20)	43	499	69,281	29	29	35	207	40,320
0	1.25	8	36	36 (26)	36	622	69,281	29	29	29	248	40,320
0	1.67	8	36	36 (50)	36	36	69,281	29	29	29	29	40,320

[a] No. of times observed.

those jobs belong at the beginning of the sequence. Therefore $\rho > 0$ and/or $K \neq 1$ should reduce, on the average, the number of nodes created and the maximum number stored. Computational experience for 8 jobs in a jobset supports this hypothesis, and, we think, establishes that the equally loaded, uncorrelated case is not a straw man.

COMPUTATIONAL RESULTS FOR $M=2$, MEAN COMPLETION TIME

THE ALGORITHM was programmed in CLP (Cornell List Processor). 50 jobsets were run for each set of conditions on ρ, K, and n. Total computer time on the CDC 1604 was 1.5 hours, ranging from about 4 minutes for the

most difficult 9-job problem to 0.75 seconds for any of the 5-job problems.

It is clear from Table I that computational effort increases sharply with the number of jobs, n. Using the median as a measure, the number of nodes created roughly doubles each time a job is added in the uncorrelated, equally loaded case. One reason could be that the solution to an $(n-1)$ job subset does not help much in solving the n job, because the order of the jobs in the optimum sequence for $(n-1)$ may not be preserved in full n job sequence. For example, suppose we have the following jobset:

i	1	2	3
a_i	2	10	1
b_i	11	3	8

The optimum sequence for jobs 1 and 2 is 1–2. But the optimum sequence for all 3 jobs is 3–2–1, which reverses the order of jobs 1 and 2.

From Table I, it is evident that as n increases, the number of nodes in the tree (= maximum possible number created) increases faster than the median number created by branch and bound. Since direct enumeration would make use of the tree structure, the above implies that branch and bound gets relatively better than enumeration as n goes up. However, for small n (say 4 or less) it is likely that direct enumeration will be easier.

For the interested reader, a difficult 9-job problem and one of average difficulty are given below. The jobs have been numbered so that 1–2–· · ·–9 is *an* optimal sequence.

	Difficult	Average
i	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
a_i	4 4 4 7 4 7 5 8 7	2 4 3 1 7 6 8 7 10
b_i	5 2 9 1 8 4 8 1 9	4 1 1 7 3 8 7 9 7
Number of nodes created:	934	321
Sum of completion times:	275	231

COMPUTATIONAL RESULTS FOR 3 MACHINES, MAKESPAN

THE FIRST problems that were solved are taken from those published in the literature. Giglio and Wagner[3] give processing times for 6 jobsets of 6 jobs each; Dudek and Teuton[4] do so for one jobset of 6 jobs. For the remaining problems that are reported, the processing times were all independent, identically distributed, rounded-down random numbers—analogous to $\rho=0$, $K=1$ in the 2-machine case. As in the 2-machine,

TABLE II

3 MACHINE, MAKESPAN—RESULTS FOR PREVIOUSLY PUBLISHED JOBSETS

Jobset	No. of jobs	Min poss. no. of nodes created	No. of nodes created	Computing time, sec.
G & W #1	6	21	40	0.65
G & W #2	6	21	21	1.00
G & W #3	6	21	55	1.50
G & W #4	6	21	21	2.40
G & W #5	6	21	87	1.75
G & W #6	6	21	152	2.00
D & T #1	6	21	21	1.60

mean completion-time case, the 3-machine makespan problem is unaffected by the location parameter of the processing times (*see* Appendix). Experience on a few problems with processing times that are correlated within jobs indicates that these problems are probably *harder* than the uncorrelated ones.

The algorithm was programmed in LPF (List Processing Fortran), a language developed at the Cornell Computing Center. Computing times are on the CDC 1604 and include the time to input the jobset and output the solution on magnetic tape. A lower bound slightly more sophisticated than the one given previously was used. In it, TIME$B(J_r)$ is replaced by max[TIME$B(J_r)$, TIME$A(J_r)+\min_{J_r} a_i$] and TIME$C(J_r)$ is replaced by max[TIME$C(J_r)$, TIME$B(J_r)+\min_{J_r} b_i$, TIME$A(J_r)+\min_{J_r}(a_i+b_i)$]. For all 6 of the Giglio and Wagner jobsets, the number of nodes created is less

TABLE III

3 MACHINE, MAKESPAN—RESULTS FOR JOBSETS GENERATED BY THE AUTHORS

No. of jobs in jobset	No. of nodes created						Computing time (sec)			
		Observed in 50 jobsets					Observed in 50 jobsets			
	Min possible	Min [a]	Overall mean	Mean those requiring more than min	Max	Max possible	Min	Overall mean	Mean those requiring more than min	Max
4	10	10 (35)	12	16	29	41	0.53	0.55	0.61	0.73
5	15	15 (33)	20	30	51	206	0.53	0.61	0.74	0.97
6	21	21 (31)	44	82	217	1,237	0.60	0.94	1.48	3.67
7	28	28 (30)	59	105	467	8,660	0.65	1.19	1.99	9.13
8	36	36 (33)	120	284	1455	69,281	0.72	4.2	10.35	50.68
9	45	45 (39)	181	661	1687	623,530	0.78	5.7	22.39	74.45
10	55	55 (38)	212	711	2570	6,235,301	0.97	8.7	32.99	150.32

[a] No. of times observed.

than half the number of iterations that G&W's best integer programming form required. Since it would seem that one calculation of a lower bound is much simpler than one integer programming iteration, branch and bound looks good.

Computation time on the IBM 1620 for the 6-job jobsets that Dudek and Teuton solved (perhaps including the one for which they give processing times) was from 0.89 to 4.25 minutes, with a mean of 2.07 minutes. The dissimilarities between the two computers make it difficult to compare this to the 0.65 to 2.40 seconds for branch and bound on the CDC 1604.

The main difference between these results and those for the 2-machine problems is that a higher percentage of these jobsets are solvable with minimum effort. Judging from the maximum number of nodes created and recalling that 'branched from' nodes are saved in the 3-machine computations, those that are not solvable with a minimum of effort require more effort than their 2-machine counterparts.

For the interested reader, the hardest 10-job problem is given below. The sequence 1–2–· · ·–10 is an optimal one for this problem.

HARDEST 10 JOB PROBLEM

i	1	2	3	4	5	6	7	8	9	10
a_i	1	5	7	8	3	7	9	8	6	3
b_i	2	9	6	9	2	10	7	9	1	1
c_i	9	7	8	9	3	4	7	4	3	1

Makespan = 66

POSSIBLE EXTENSIONS

AS LITTLE ET AL. have mentioned, the branch-and-bound technique is a method of wide applicability. In the 2-machine problem, a variety of objective functions could be handled. For example, lower bounds for a weighted sum of completion times or mean one-sided lateness or other criteria could be constructed, and the branch-and-bound technique applied.

Other generalizations of the scheduling problem can also be handled. When there are more than 3-machines, minimum makespan schedules may require that the jobs be done in different sequences on different machines.* For example, for 4 machines, there will be two sequences (possibly identical), one for the first 2 machines, the other for the last 2. There are now $(n!)^2$ possible schedules, but lower bounds can be obtained and the branch-and-bound technique applied.

It is not only possible, but probable that the two sequences will be

* For 4 machines with $a_1=d_1=b_2=c_2=4$, $b_1=c_1=a_2=d_2=1$, the minimum makespan schedule has job 2 precede 1 on machines A and B, and job 1 precede 2 on machines C and D. Also *see* JOHNSON.[5]

identical, and good schedules can be obtained by restricting attention to schedules where the jobs are processed in the same order on all 4 machines. Straightforward extension of the 3-machine lower bounds permits the use of the branch-and-bound technique on this restricted problem. Extension to more than 4 machines, still maintaining the restriction of a common sequence on all machines, is also possible.

Brooks and White[3] consider jobsets for which the flow restriction has been removed. They proceed directly up the tree, using lower bounds to determine which path to follow, and obtain good, but not necessarily optimal, schedules for up to 18 jobs on 10 machines.

APPENDIX

The following theorem shows that, for the 2-machine mean completion-time or 3-machine makespan problems, changing the location and/or the scale of the processing times does not affect the solution.

Theorem. *If the processing times for two flow-shop scheduling problems $\tilde{P}$ and P' are related by*

$$\tilde{a}_i = H(a_i' + G),$$
$$\tilde{b}_i = H(b_i' + G),$$
$$\tilde{c}_i = H(c_i' + G) \quad \textit{for some} \quad H, G > 0 \quad \textit{and for every} \quad i,$$

then (1) the same sequence is optimal for both problems and (2) the branch-and-bound technique will follow the same path in finding that sequence for both problems.

Proof. Let us first assume that $H = 1$ and show that the value of G makes no difference.

Consider any schedule in which jobs are processed in the same order by all M machines. The effect of adding a constant G to all processing times is easily seen to be an increase in the completion time of the rth job in the schedule by $G(r+M-1)$. Therefore, in the 2-machine case, the sum of the completion times will be increased by $\sum_{r=1}^{r=n} G(r+1) = \frac{1}{2} Gn(n+3)$. The same is true for the lower bounds: both $\hat{T}_r$ and $\hat{S}_r$ are increased by $G[r(n-r) + \frac{1}{2}(n-r)(n-r+1) + (n-r)]$ and $\sum_{J_r} d_i$ is increased by $\frac{1}{2} r(r+3)$, so $LB(J_r)$ is increased by their sum, which is $\frac{1}{2} Gn(n+3)$. The same arguments hold for the 3-machine case, where all lower bounds and makespans are increased by $G(n+2)$.

Since the same constant is added to the value of all sequences, the optimality of a sequence is unaffected by the value of G. Since the same constant is added to all the lower bounds, the ranking of the lower bounds is unaffected by the value of G, implying that the path taken by the branch-and-bound procedure is also unaffected. The conclusion is that the location parameter of the processing times makes no difference.

Now suppose $H \neq 1$. Define new time units so that when $\tilde{a}_i$, $\tilde{b}_i$, and $\tilde{c}_i$ are measured in these units, and a_i', b_i', and c_i' in their original units, H will be 1. So for problem $\tilde{P}$ in new time units, the same path to the same solution will be taken as for P'. But an optimal sequence for $\tilde{P}$ in new time units is also one for $\tilde{P}$ in original units. Therefore, H makes no difference.

REFERENCES

1. J. D. C. Little, K. G. Murty, D. W. Sweeney, and C. Karel, "An Algorithm for the Traveling Salesman Problem," *Opns. Res.* **11,** 972–89 (1963).
2. A. H. Land and A. Doig, "An Automatic Method of Solving Discrete Programming Problems," *Econometrica* **28,** 497–520 (1960).
3. R. J. Giglio and H. M. Wagner, "Approximate Solutions to the Three-Machine Scheduling Problem," *Opns. Res.* **12,** 305–19 (1964).
4. R. A. Dudek and O. F. Teuton, Jr., "Development of M-Stage Decision Rule for Scheduling n Jobs Through m Machines," *Opns. Res* **12,** 471–97 (1964).
5. S. M. Johnson, "Optimal Two- and Three-Stage Production Schedules with Setup Times Included," *Naval Res. Log. Quart.* **1,** 61–8 (1954); *also* reprinted as Chap. 2 of J. F. Muth and G. L. Thompson, *Industrial Scheduling,* Prentice-Hall, 1963.
6. R. W. Conway, J. J. Delfausse, W. L. Maxwell, and W. E. Walker, "CLP—The Cornell List Processor," to appear in *Communications of the Association for Computing Machinery.*
7. D. Bessel, "LPF, List Processing Fortran," Reference Manual, Cornell University, April 1964.
8. G. H. Brooks and C. White, "An Algorithm for Finding Optimal or Near Optimal Solutions to the Production Scheduling Problem," *J. Indust. Eng.* **16,** 34–40 (1965).