Exact, Approximate, and Guaranteed Accuracy Algorithms for the Flow-Shop Problem $n/2/F/\bar{F}$

WALTER H. KOHLER

University of Massachusetts, Amherst, Massachusetts

AND

KENNETH STEIGLITZ

Princeton University, Princeton, New Jersey

ABSTRACT. Improved exact and approximate algorithms for the n-job two-machine mean finishing time flow-shop problem, $n/2/F/\bar{F}$, are presented While other researchers have used a variety of approximate methods to generate suboptimal solutions and branch-and-bound algorithms to generate exact solutions to sequencing problems, this work demonstrates the computational effectiveness of coupling the two methods to generate solutions with a guaranteed accuracy. The computational requirements of exact, approximate, and guaranteed accuracy algorithms are compared experimentally on a set of test problems ranging in size from 10 to 50 jobs The approach is readily applicable to other sequencing problems

KEY WORDS AND PHRASES. sequencing problem, algorithms, local search, branch-and-bound, guaranteed accuracy, computational results

CR CATEGORIES: 5 25, 5.39, 5 49

1. *Introduction*

This paper presents improved exact and approximate algorithms for solving the classic n-job two-machine mean finishing time flow-shop problem, $n/2/F/\bar{F}$ [2]. In this sequencing problem, each job is assumed to consist of at most two distinct operations, where each operation can be performed on only one of two distinct machines. The processing time required by job i on machine j will be denoted by p_{ij}. In keeping with the flow-shop assumption that the operations of each job are processed by the machines in the same order, we will assume that each job completes its processing on machine 1 before it begins processing on machine 2. Permutation schedules result when each machine processes the jobs in the same order (permutation), with no unnecessary idle time between operations. When the mean of the job finishing times is used as the measure of schedule performance, it is well known that the class of permutation schedules is guaranteed to contain an optimal solution [2]. Although the mean finishing time criterion has not received as much attention as the makespan (maximum job finishing time) criterion [1–5, 11], Ignall and Schrage [6] presented computational results for an exact branch-and-bound approach and

This work was supported by the NSF under grant NSF-GJ-965, and the US Army Research Office-Durham under contract DAHCO4-69-C-0012.

Authors' addresses W H Kohler, Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA 01002; K. Steiglitz, Department of Electrical Engineering, Princeton University, Princeton, NJ 08540.

Krone and Steiglitz [9] applied local search techniques to the general m-machine case $n/m/F/\bar{F}$. We will present improved exact and approximate algorithms for the two-machine problem and demonstrate the computational effectiveness of coupling local search and branch-and-bound to generate solutions with a guaranteed accuracy.

2. *Branch-and-Bound Exact Algorithms*

Exact branch-and-bound algorithms for combinatorial optimization problems whose solution space is the set of permutations of n objects have been characterized in our recent paper [8] by the sextuple (B_p,S,E,D,L,U), where

B_p is the branching rule for permutation problems (fixed),
S is the next node selection rule;
E is the set of node elimination rules;
D is the node dominance relation;
L is the node lower-bound cost function;
U is an upper-bound solution cost.

We refer the reader to that paper for precise definitions of these terms and an explicit description of a general class of branch-and-bound algorithms.

Here we will be concerned with the following choices of parameters:

1. $S = LLB_{FIFO}$ (*l*east-*l*ower-*b*ound). Select the currently active node with the least lower-bound cost. In the case of ties, select the node that was generated first.

2. $E \subseteq \{U/DBAS, U/DB, AS/DB\}$.

(a) *U/DBAS* (*u*pper-bound tested for dominance of *d*escendants of the *b*ranching node and members of the *a*ctive *s*et). Eliminate those descendants of the branching node and those members of the active set whose lower-bound costs exceed the current upper bound U.

(b) *U/DB* (*u*pper-bound tested for dominance of *d*escendants of the *b*ranching node). Eliminate those descendants of the branching node whose lower-bound costs exceed the current upper bound U.

(c) *AS/DB* (*a*ctive node *s*et tested for dominance of *d*escendants of the *b*ranching node). Eliminate those descendants of the branching node that are dominated by a member of the active set.

3. $D = D_{IS}$ (*I*gnall and *S*chrage [6]). Let π_y and π_z be two nodes which represent partial schedules involving the same subset of jobs, P, and let $F_{ij}(\pi)$ be the finishing time of job i on machine j under the partial schedule π. Then $(\pi_y, \pi_z) \in D_{IS}$ (that is, π_y dominates π_z) if and only if (1) $\max_{i\in P} F_{i2}(\pi_y) \leq \max_{i\in P} F_{i2}(\pi_z)$, and (2) $\sum_{i\in P} F_{i2}(\pi_y) \leq \sum_{i\in P} F_{i2}(\pi_z)$. Note that $\max_{i\in P} F_{i1}(\pi_y) = \max_{i\in P} F_{i1}(\pi_z) = \sum_{i\in P} p_{i1}$, since there is no idle time between operations on the first machine.

4. $L = L_{IS}$ (*I*gnall and *S*chrage [6]). This is the lower-bound function of Ignall and Schrage [6]. The reader is referred to their paper for details.

5. Let $U(\pi_b)$ denote the least upper-bound solution cost known at the time π_b is the current branching node. Then $U(e)$ denotes the initial upper bound that appears in the sextuple specification of each algorithm. Here we will consider two possibilities:

(a) $U(e) = \infty$, when no initial solution is given,

(b) $U(e) < \infty$, when some initial solution is given.

3. *Approximate Algorithms*

A. Nonbacktracking Branch-and-Bound. A tree search with no backtracking can be used to quickly generate a complete solution, whose cost then is an upper bound on the optimal cost. Such an algorithm results from choosing the following parameters: $S = LLB_{FIFO}$, $E = \varnothing$, $D = \varnothing$, $L = L_{IS}$, $U(e) = \infty$, and limiting the number of active nodes to one. In this case the least lower-bound descendant at each branching node becomes the next branching node. This algorithm will be denoted by $BBLB(1)$, and is similar to one described by Ashour [1].

B. Local Neighborhood Search. The local search approach to finding approximate solutions to combinatorial problems has been described by many authors [10, 12, 13]. Its specific application to the flow-shop problem can be found in [7, 9]. The idea of such algorithms is to search from one solution in some neighborhood defined by a local transformation, adopting improvements as they are found, and continuing this process until no further local improvement is possible. This is possibly repeated from many different random starting solutions, and the best solution retained. This class of algorithms will be denoted by $LNS(I, P, N)$ (*l*ocal *n*eighborhood *s*earch), where (1) I is the method for choosing initial solutions, (2) P is the policy for searching the neighborhood and accepting improvements, and (3) N is the neighborhood searched for improvements.

We will be concerned here with the following choices:

1. (a) $I = UPRS$ (*u*niform *p*seudorandom *s*tart), in which starting solutions are generated using random permutations.

(b) $I = BBLB(1)$, in which the single result of algorithm $BBLB(1)$ is used as the start.

2. $P = RFI$ (*r*andom *f*irst *i*mprovement), in which the neighborhood is enumerated randomly and the first improvement encountered is accepted.

3. (a) $N = BSI$ (*b*ackward *s*ingle *i*nsertion) is the set of permutations (solutions) generated by inserting the job currently in position i after the job currently in position j, $i < j$. For example, $i = 2$, $j = 4$ transforms solution *abcde* into solution *acdbe*.

(b) $N = FSI$ (*f*orward *s*ingle *i*nsertion) is the set of permutations generated by inserting the job currently in position i before the job currently in position j, $i > j$. For example, $i = 4$, $j = 2$ transforms solution *abcde* into solution *adbce*.

(c) $N = API2$ (*a*djacent *p*air *i*nterchange 2) is the set of permutations generated by first interchanging the jobs in positions i and $i + 1$, and then, if a second number $j \neq i$ is specified, interchanging the jobs then in positions j and $j + 1$. For example, $i = 2$, $j = 4$, transforms *abcde* into *acbed*. Also, $i = 2$, $j = 3$ transforms *abcde* into *acdbe*.

(d) $N = BSI \cup API2$.

All of these neighborhoods are of order n^2 in size, denoted $O(n^2)$.

4. *Computational Results*

A. Problem Data. Twenty-five standard sets of random data, five sets for each of the problem sizes $n = 5$, 10, 15, 20, and 50, were used to test exact and approximate algorithms. The processing times, p_{ij}, were selected as independent integer samples from the uniform distribution $\text{prob}\{p_{ij} = r\} = \frac{1}{10}$ for $r = 1, \cdots, 10$. Rather than computing the mean finishing time, which may not be integer valued, the sum of the finishing times was adopted as an equivalent measure of schedule cost.

B. Evaluation of Approximate Techniques. Local neighborhood search and nonbacktracking branch-and-bound were used to generate suboptimal solutions to the twenty-five test problems. In the first set of experiments, the local neighborhood search algorithm was used with parameters $I = UPRS$; $P = RFI$; and the neighborhoods $N = BSI$, FSI, $API2$, and $BSI \cup API2$.

Table I shows the best solution produced by each neighborhood and indicates the number of times it was produced and the number of starting solutions used, i.e. # = number of times best solution found/number of starting solutions used. The starred solutions have been proved optimal by an exact technique and the underlined solutions are the best known suboptimal solutions for those data sets. Problems of size $n = 5$ have not been included since they are relatively simple problems and all neighborhoods frequently produced the optimal solution for each data set. A comparison of the solution quality based on this small number of data sets shows no readily apparent differences between neighborhoods. No one neighborhood was uniformly better than any other on all data sets. In particular, it is interesting to note that $BSI \cup API2$ is not uniformly better than BSI or $API2$.

Branch-and-bound without backtracking, $BBLB(1)$, was also employed to generate

TABLE I BEST SOLUTIONS GENERATED BY APPROXIMATE TECHNIQUES

n	Data Set	Optimal Solution* Best Known Solution	LNS(PRS,RFI,N)								BBLB(1)	LNS(BBLB(1),RFI,N)			
			N=BSI		N=FSI		$N=(API)^2$		N=BSI $(API)^2$			N=BSI		N=FSI	
			Best Solution	#	Best Solution	#	Best Solution	#	Best Solution	#	Solution	Best Solution	#	Best Solution	#
10	1	298*	299	2/10	298*	3/10	304	2/10	299	1/10	305	298*	6/10	298*	5/10
	2	326*	326*	3/10	326*	7/10	326*	5/10	326*	5/10	335	326*	9/10	326*	10/10
	3	375*	375*	4/10	375*	6/10	375*	3/10	375*	3/10	375*	375*	1/1	375*	1/1
	4	340*	340*	1/10	340*	5/10	340*	1/10	340*	5/10	340*	340*	1/1	340*	1/1
	5	291*	291*	1/10	291*	8/10	291*	4/10	291*	3/10	294	291*	1/10	291*	10/10
15	1	598	605	2/10	599	1/10	602	3/10	598	1/10	667	603	3/10	600	5/10
	2	608*	608*	6/10	608*	8/10	608*	9/10	608*	8/10	609	608*	10/10	608*	10/10
	3	573	576	1/10	573	1/10	574	1/10	577	1/10	600	578	1/10	578	6/10
	4	713*	713*	8/10	713*	7/10	713*	7/10	713*	8/10	713*	713*	1/1	713*	1/1
	5	634*	639	1/10	636	1/10	642	1/10	637	1/10	638	638	1/1	638	1/1
20	1	954	971	1/10	956	1/10	958	1/10	955	1/10	956	956	1/1	954	10/10
	2	1161	1166	1/10	1173	1/10	1177	1/10	1170	2/10	1310	1161	1/10	1170	1/10
	3	957	958	2/10	958	1/10	962	1/10	957	1/10	991	957	1/10	958	4/10
	4	1015	1020	1/10	1018	1/10	1019	1/10	1016	1/10	1041	1015	1/10	1019	5/10
	5	881*	885	1/10	886	1/10	884	1/10	890	2/10	886	885	10/10	885	10/10
50	1	6295	6334	1/5	6327	1/5					7127	6295	1/5	6324	1/5
	2	5046	5151	1/5	5079	1/5					5807	5046	1/5	5068	1/5
	3	6300	6392	1/5	6455	1/5					6990	6300	1/5	6409	1/5
	4	6362	6519	1/5	6460	1/5					6389	6362	1/5	6382	1/5
	5	5726	5927	1/5	5949	1/3					6017	5726	1/5	5805	1/5

suboptimal solutions. The solutions generated by this heuristic were generally not as good as the local neighborhood search solutions, but they required less than half the computation time. The next step was to use these solutions as improved nonrandom starting solutions for local neighborhood search. *LNS*(*BBLB*(1), *RFI*, *N*) was tested with two neighborhood sets $N = BSI$ and *FSI*. For small problems ($n \leq 15$) we found that the solutions generated with these improved starting solutions were about the same as with the random starting solutions, but the improved start usually generated better solutions for the large problems ($n = 20, 50$). Also, the *BSI* neighborhood produced uniformly better solutions than *FSI* for the 50-job problems, but about the same quality for the smaller problems. The solutions are tabulated in Table I along with the optimal or best known solutions for each data set.

When $n \leq 15$, the average time required to generate a locally optimal solution using *LNS*(*UPRS*, *RFI*, *BSI*) was greater than the sum of the times required by *BBLB*(1) and *LNS*(*BBLB*(1), *RFI*, *BSI*). This suggests that a good heuristic for large mean finishing time flow-shop problems is to generate an improved starting solution using nonbacktracking branch-and-bound and then to try to improve the solution with local neighborhood search.

The average number of neighbors that are tested (selected and cost compared to base solution cost) before finding a local optimum can be used as a relative measure of execution time requirements for local search. This also measures the number of cost function evaluations performed For n-job m-machine flow-shop problems, each cost evaluation requires time that grows linearly with the product nm, since the finishing times of each of n jobs on m machines must be computed by recursively building the permutation schedule until all n jobs have been assigned a starting time on each of m machines. Experimental results for our five problem sets indicate that the number of neighbors tested using *LNS*(*UPRS*, *RFI*, *N*) and *LNS*(*BBLB*(1), *RFI*, *N*) grows as $O(n^{2.3}) - O(n^{2.5})$, and the execution time as $O(n^{3.3}) - O(n^{3.7})$, when $N = BSI, FSI, API2$, and $BSI \cup API2$ Thus, the number of neighbors tested seems to grow slightly faster than the neighborhood size $O(n^2)$. Nonbacktracking branch-and-bound always generates $n(n-1)/2 + 1$ nodes, each requiring a lower-bound function evaluation; and consequently, the execution time increases as approximately $O(n^3)$.

C. Evaluation of Exact and Guaranteed Algorithms. The branch-and-bound approach used by Ignall and Schrage [6] for the two-machine mean finishing time problem is equivalent to BB_2 to be described below, except that they use LLB_{LIFO}, breaking ties in the selection rule with a last-in-first-out policy, instead of our first-in-first-out policy. This change was found to have a negligible effect on the experimental results to be described. From the analysis in [8] we know that the computational requirements of this algorithm may be reduced, and cannot be increased, by adding upper-bound elimination rule *U/DBAS* to *E*. In addition it was shown that the computational requirements are a monotone nonincreasing function of the initial upper-bound cost $U(e)$. The analysis in [8] is valid for any measure of computation that is a monotone nondecreasing function of (1) the total number of nodes generated, (2) the total number of nodes actually branched from, and (3) the size of the branched-from and active node sets at each stage of the algorithm. Measures based on these statistics are independent of the data structures, language, or computer used to implement the algorithm. Consequently, in addition to the measured CPU time, we have tabulated data on the maximum number of active nodes (a good measure of the storage required) and the total number of nodes generated (a very rough measure of the relative time required). The next set of computational results will show that the average computational requirements for the mean finishing time flow-shop problem can be significantly reduced by adding a slightly weaker version of rule *U/DBAS* to *E* and using a good suboptimal solution as the initial upper bound $U(e)$.

In our computational experiments, we assumed that the upper-bound cost at each branching node would be used only to eliminate the descendants of that branching node

and not the currently active set. This new rule is denoted by *U/DB*. *U/DB* is weaker than *U/DBAS* since an active node π_a not eliminated by the upper bound when generated, i.e. $L(\pi_a) \not> U(P(\pi_a))$, where $P(\pi_a)$ denotes the immediate ancestor of π_a, may have $L(\pi_a) > U(\pi_b)$ at a later branching node π_b if an improved upper-bound solution was found. By not rechecking for possible dominance of the active set when the upper bound is improved, some unnecessary nodes may be on the active list and may be branched from. However, in our experiments, the initial upper-bound $U(e)$ was usually an optimal solution and consequently *U/DB* and *U/DBAS* would eliminate the same nodes. Furthermore, since we used the *LLB* selection rule exclusively, the set of branched-from nodes cannot include nodes with lower bound greater than the optimum cost ([8, Lem. 6]). Therefore, using *U/DB* rather than *U/DBAS* did not change the branched-from set. Now consider

$$BB_2 = (B_p, LLB_{FIFO}, \{AS/DB\}, D_{1S}, L_{1S}, U_2(e) = \infty), \quad \text{and}$$

$$BB_1 = (B_p, LLB_{FIFO}, \{AS/DB, U/DB\}, D_{1S}, L_{1S}, U_1(e)),$$

where $U_1(e)$ = best solution generated by *LNS*(*BBLB*(1), *RFI*, *BSI*). BB_1 differs from BB_2 by the addition of rule *U/DB* to the set of elimination rules and the use of a soboptimal solution as the initial upper bound. The computational requirements of BB_2 and BB_1 are displayed in Table II. *T* denotes the rule by which the algorithm terminated for each data set· R1 ≙ Rule 1 for an optimal solution [8] (the upper-bound solution proved optimal); R2 ≙ Rule 2 for an optimal solution [8] (the next branching node is a complete solution), *ET* ≙ execution time equals or exceeds prespecified limit, *EN* ≙ number of currently active nodes equals maximum active node limit. When termination occurs under *ET* or *EN*, the cost of the least lower-bound active node is listed as the final lower-bound cost. When termination occurs under R1 or R2, the final lower-bound cost is the cost of an optimal solution π^*.

BB_1 and BB_2 both terminated with optimal solutions (R1) for all problems of size $n = 10$, but an examination of Table II shows that the computational requirements of BB_1 are significantly less than BB_2. The maximum-number-of-active-nodes statistic can be interpreted as the minimum storage needed to execute the algorithm. An average storage requirement for problems of a given size n was found by averaging this statistic. When $n = 10$ the average storage requirement for BB_1 was only 15 percent of the average storage requirement for BB_2; and the average measured execution time used for BB_1 was only 13 percent of that used for BB_2. When the time required to generate the initial upper-bound solution is included (the time required by *LNS*(*BBLB*(1), *RFI*, *BSI*) for ten starts per data set), there is still an average savings of 76 percent. The improvement in execution time and storage occurs because all descendants with lower-bound costs greater than upper-bound cost *U* are immediately eliminated by rule *U/DB*. These descendants never become active and it is unnecessary to test if they are dominated by existing active nodes under rule *AS/DB*.

When the problem size was greater than ten, our preset storage and execution time limits were frequently exceeded. However, since the computational requirements of BB_1 are less than or equal to the requirements of BB_2, BB_1 generally got closer to an optimal solution before termination. In some cases (problems 15-5 and 20-5), BB_1 reached an optimal solution while BB_2 exceeded the computational limits. When both algorithms are terminated for exceeding storage or execution time limits, the final lower bound achieved by BB_1 is at least as great and frequently greater than the bound achieved by BB_2. Because we are using the *LLB* branching rule, the active node with the least lower-bound cost is always the current branching node. This lower-bound cost, denoted by *LB*, together with the final upper bound *U* can be used to bracket the cost of an optimal solution. This feature has not been exploited in the computational work of other researchers. Table II indicates brackets $(U\text{-}LB)/U$ of 0.–2.3 percent for problems of size $n = 15$,

TABLE II COMPUTATIONAL REQUIREMENTS OF EXACT AND GUARANTEED ACCURACY TECHNIQUE

n	Data Set	BB_2					BB_1							BB_1 : r = .95(n = 10,15, 20), .90(n = 50)						
		T	Final Lower-Bound	Maximum #Active Nodes	Total #Nodes Generated	Time* (seconds)	Initial Upper-Bound	T	Final Lower-Bound	Maximum #Active Nodes	Total #Nodes Gen.	Time* (seconds)	$\frac{U-LB}{U}$	Initial Upper-Bound	T	Final Lower-Bound	Maximum #Active Nodes	Total #Nodes Gen.	Time* (seconds)	$\frac{U-LB}{U}$
10	1	R1	298	381	724	2.787	298	R1	298	50	622	.319	.0	298	AB	285	33	75	.046	.045
	2	R1	326	132	178	.256	326	R1	326	12	109	.048	.0	326	AB	320	4	11	.013	.018
	3	R1	375	61	83	.060	375	R1	375	4	64	.030	.0	375	AB	367	1	11	.010	.021
	4	R1	340	54	69	.045	340	R1	340	2	43	.022	.0	340	AB	339	1	11	.011	.003
	5	R1	291	456	1579	8.079	291	R1	291	95	1366	1.026	.0	291	AB	282	6	11	.011	.031
15	1	EN	580	999	1819	17.062	603	ET	589	942	5726	30.072	.023	603	AB	573	242	553	1.221	.050
	2	R1	608	237	291	.743	608	R1	608	11	168	0.089	.0	608	AB	606	2	16	.016	.033
	3	EN	563	999	1716	16.829	578	EN	567	999	4263	29.856	.019	578	AB	550	21	30	.026	.048
	4	R1	713	428	738	3.073	713	R1	713	29	497	.264	.0	713	AB	709	4	16	.016	.006
	5	EN	629	999	2242	22.271	638	R1	634	167	4605	3.872	.0	638	AB	607	49	153	.128	049
20	1	EN	923	999	2494	26.631	956	ET	943	893	16524	40.026	.014	956	AB	909	87	299	.320	.049
	2	EN	1122	999	1495	15.764	1161	EN	1122	999	1738	15.842	.035	1161	AB	1111	17	21	.024	.043
	3	EN	940	999	1887	22.160	957	EN	944	999	6656	32.531	.014	957	AB	935	10	21	.023	.023
	4	EN	976	999	1650	16.802	1015	EN	980	999	3244	23.156	.036	1015	AB	968	8	21	.022	.046
	5	EN	876	999	2462	27.367	885	R1	881	124	5415	4.436	.0	885	AB	858	4	21	.022	.031
50	1	EN	5842	999	1130	10.882	6295	EN	5842	999	1130	10.635	.065	6295	AB	5835	50	51	.114	.073
	2	EN	4716	999	1050	9.507	5046	EN	4716	999	1050	9.260	.065	5046	AB	4716	50	51	.117	.065
	3	EN	5795	999	1070	10.516	6300	EN	5795	999	1070	10.017	.080	6300	AB	5788	50	51	.117	.081
	4	EN	5945	999	1046	10.304	6362	EN	5946	999	1079	9.920	.050	6362	AB	5925	43	51	.106	.069
	5	EN	5050	999	1136	10.390	5726	EN	5050	999	1136	9.855	.118	5726	EN	5050	999	1136	10.330	.118

*On the IBM 360/91, FORTRAN H compiler.

0.–3.6 percent for problems of size $n = 20$, and 5.0–11.8 percent for problems of size $n = 50$.

These experiments suggest that a good suboptimal solution should be obtained before attempting to generate an optimal solution with branch-and-bound. The use of elimination rule U/DB together with a good initial upper-bound solution significantly reduces the necessary computation, and if the branch-and-bound algorithm exceeds the allowed storage or execution time limits, the known upper-bound solution and the least lower-bound active node nevertheless give a closs bracket on the cost of an optimal solution.

D. Bracketing with Suboptimal Solutions. When an optimal solution is not necessary, even more dramatic computational savings are possible. In this section we use the branch-and-bound algorithm to verify that a suboptimal solution exceeds the optimal cost by no more than a prespecified amount. This technique for achieving a prespecified bracket has been suggested by other authors and is discussed in [7]. We will again be using BB_1, but now given prespecified accuracy ratio r, $0 < r \leq 1$, BB_1 will be terminated at branching node π_b if $r \cdot U(\pi_b) \leq L(\pi_b)$. The cost $f(\pi^*)$ of an optimal solution π^* is then bracketed by $r\ U(\pi_b) \leq L(\pi_b) \leq f(\pi^*) \leq U(\pi_b)$. The case when $r = 1$ was discussed in the previous section. Table II shows the computational requirements for BB_1 when $r = .95$ ($n = 10$, 15, and 20) and $r = .90$ ($n = 50$). $T = AB$ signifies that BB_1 achieved the prespecified bracket. In most cases the bracket was achieved at the first branching step (the total number of nodes generated is then $n + 1$) and the required execution time was insignificant compared to the optimum producing branch-and-bound requirements. This is a result of the fact that the initial upper-bound solutions are optimal and the lower-bound function L generates a very close bound. When the bracket was tightened to $r = .99$, the average computational requirements were not significantly different from the optimal case, since both methods usually exceeded the allowed computational limits. However, on data sets where this was not the case, savings were achieved.

5. *Conclusions*

The computational results with the $n/2/F/\bar{F}$ flow-shop problem indicate that computational requirements for optimum producing branch-and-bound algorithms can be decreased by using a good initial upper-bound solution together with an upper-bound dominance rule. The reductions are large enough so that the total computation (the computation required to obtain an initial upper-bound solution, plus the computation required by the optimum producing branch-and-bound algorithm) may be significantly reduced by first computing a good suboptimal solution. Use of a good upper bound with optimum producing branch-and-bound is computationally competitive with approximate algorithms for problems as large as $n = 20$.

When a suboptimal solution guaranteed to satisfy a prespecified bracket is sufficient, the branch-and-bound technique can be used to verify the quality of a suboptimal solution or to generate another suboptimal solution satisfying the desired bracket. Brackets of 5–10 percent frequently result in dramatic computational savings over the optimum producing (0 percent) requirements. Applying local neighborhood search to the solution obtained from branch-and-bound without backtracking provides a good initial upper-bound solution.

REFERENCES

1. Ashour, S. An experimental investigation and comparative evaluation of flow-shop scheduling techniques *Oper Res 18*, 3 (May–June 1970), 541–549
2. Conway, R W., Maxwell, W L , and Miller, L W. *Theory of Scheduling*. Addison-Wesley, Reading, Mass., 1967
3. Day, J E , and Hottenstein, M P Review of sequencing research. *Naval Res Logistics Quart 17*, 1 (March 1970), 11–40
4. Elmaghraby, S E The machine sequencing problem-review and extensions *Naval Res Logistics Quart 15*, 2 (June 1968), 205–232.

5. GUPTA, J N. D. Economic aspects of production scheduling systems *J. Oper Res Soc Japan 13*, 4 (March 1971), 169–193
6 IGNALL, E., AND SCHRAGE, L Application of the branch and bound technique to some flow-shop sequencing problems *Oper. Res 13*, 3 (May–June 1965), 400–412.
7. KOHLER, W. H Exact and approximate algorithms for permutation problems Ph.D. Diss , Dep of Elec. Eng., Princeton U , Princeton, N J., Aug 1972
8 KOHLER, W. H , AND STEIGLITZ, K. Characterization and theoretical comparison of branch-and-bound algorithms for permutation problems. *J ACM 21*, 1 (Jan 1974), 140–156
9. KRONE, M., AND STEIGLITZ, K. Heuristic programming solution of a flowshop scheduling problem. *Oper. Res.* (to appear)
10 LIN, S Computer solutions to the Traveling Salesman Problem *Bell Syst Tech J 44*, 10 (Dec 1965), 2245–2269
11 MELLOR, P. A review of job shop scheduling *Oper Res Quart 17*, 2 (June 1966), 161–171
12 REITER, S., AND SHERMAN, G Discrete optimizing *J SIAM 13*, 3 (Sept 1965), 864–889.
13 ROTH, R. H An approach to solving linear discrete optimization problems *J. ACM 17*, 2 (April 1970), 303–313

RECEIVED MAY 1973; REVISED JANUARY 1974