New Heuristics for the Flow Line Problem with Setup Costs

Roger Z. Ríos-Mercado* and Jonathan F. Bard†

Abstract

In this work we present two heuristics for the flowshop machine scheduling problem with setup costs and makespan minimization criteria. One of the proposed procedures is an extension of an algorithm that has been very successful for the general flowshop scheduling problem. The other is a greedy randomized adaptive search procedure (GRASP) which is a technique that has successfully addressed many kinds of combinatorial optimization problems. Both procedures are compared to a previously developed algorithm. In addition, a postprocessing phase for improving the quality of the solutions is developed and adapted to each of the heuristics. All procedures are compared for two different classes of randomly generated instances. It is observed that for the case where both processing times and setup times are identically distributed, the existing heuristic proves superior to the proposed approaches; for the case where setup times are an order of magnitude smaller than the processing times, the proposed procedures outperform the existing heuristic.

1 Introduction

In many manufacturing environments such as those arising in the chemical, pharmaceutical, and processing industries, it is common for facilities or work centers to carry out several types of tasks. In this case, the use of a single system to produce, for example, different chemical compounds may require some cleansing between the process runs, and the time to set up a facility for the next task may be strongly dependent on the immediate predeccesor. Thus, a model handling this sequence-dependent property becomes crucial in facing the problem.

Sequence-dependent properties appear to be a relevant factor in other fields as well. For example, the scheduling of aircraft approaching or leaving a terminal area can be modeled as

*Ph.D. Candidate, Graduate Program in Operations Research/Industrial Engineering, Dept. of Mechanical Engineering, U. of Texas at Austin, Austin TX 78712

†Professor, Graduate Program in Operations Research/Industrial Engineering, Dept. of Mechanical Engineering, U. of Texas at Austin, Austin TX 78712

a single-machine scheduling problem. Because the time separations between successive aircraft belonging to different categories are to be changed according to their respective position, sequence-dependent processing times must be allowed for a more realistic description of the problem [3]. Other examples of situations where sequence-dependent setup times are encountered include: container manufacturing industry, where the machines are to be adjusted whenever the dimensions of the containers are changed; the paint industry, in which paints of different colors are produced on the same piece of equipment; etc.

In this article we address the problem of finding a schedule of n jobs in a m-machine flow line environment with sequence-dependent setup times that minimizes the maximum completion time $C_{\max}$ of all jobs, also known as the makespan. This problem is regarded in the scheduling literature as the sequence-dependent setup time flowshop or SDST flowshop.

The SDST flowshop is a "hard" problem in the sense that attempts to obtain an optimal solution to moderate size instances might take an exponential amount of time. Most of the real-world problems fall in this category. This becomes a critical issue since practioners and decision-making people require "quick" solutions to their problems. A typical approach is to develop heuristics; that is, algorithms that run "quickly" and intelligently deliver an approximate solution to the problem at hand. As we give up optimality in trade for quickness, the challenge is to exploit the structure of the problem that would allow us to find "good" solutions.

Our work includes the development of two new heuristics. One of the proposed heuristics is based on Nawaz et al. [11] who have been very succesful on the general flowshop scheduling problem (with no setup costs). Here, we extend their procedure to handle setup times. The other proposed algorithm is a greedy randomized adaptive search procedure (GRASP) which is a technique that has been successfully applied to several types of combinatorial optimization problems. The proposed procedures are compared to a previously developed algorithm (SETUP) due to Simons [16]. In addition, a postprocessing phase (local search) is developed and adapted to each heuristic to improve the quality of the solution. All the procedures are evaluated for two different classes of randomly generated instances. We observe that for the case where both processing and setup times are identically distributed, Simons' heuristic outperforms the proposed algorithms. For the case where the setup times are an order of magnitude smaller than the processing times, the proposed algorithms prove superior to the existing heuristic.

In Section 2 we review the main previous research related to our work. In Section 3 we formulate the problem formally and introduce some notation. Proposed heuristics and a the postprocessing phase are described in Sections 4 and 5, respectively. We then present our computational experience and results in Section 6.

2 Related Work

Research on the field of sequencing and scheduling is very broad. For excellent surveys on machine scheduling including complexity results, optimization schemes, and approximation algorithms, the reader is referred to Lawler et al. [9]. In particular, Ríos-Mercado and Bard [14] overview the main contributions on flowshop scheduling research including some applications to flexible manufacturing systems.

2.1 Heuristics for Flowshop Scheduling

Evaluation of heuristics for the makespan minimization of general flowshop scheduling problems has been very intense over the past few years. Heuristic approaches to flowshop scheduling can be divided into: (a) quick procedures: as proposed by Dannenbring [2], and Nawaz et al. [11]; (b) extensive search procedures (including techniques such as tabu search and genetic algorithms): as proposed by Widmer and Hertz [19] and Glass and Potts [7].

Several studies have shown (e.g., [18]) that the most effective procedure is the heuristic due to Nawaz et al. [11]. We attempt to take advantage of this result and extend their algorithm to the case where setup times are included. This procedure is further described in Section 4.1. The reader should bear in mind that extensive search procedures might in fact deliver a better value on the makespan, but at a cost of higher CPU time computation.

2.2 Flowshop Scheduling with Setups

The special case of a flowshop problem where setup times are considered becomes a more difficult problem. Although applications exist where setups are a big factor, very little work has been done in this area. Due to the difficulty of the problem, the researcher has to make further assumptions about the nature of the setups in an attempt to simplify the problem and develop specialized algorithms for the problem at hand. However, the resulting approaches become useless for handling problems where the nature of the setups arise from a different interaction.

Single- and two-machine flowshop scheduling with setups is discussed in our survey [14]. To the best of our knowledge, Simons [16] has done the only work on heuristics for the flowshop scheduling problem with sequence-dependent setup times. His algorithm (SETUP) transforms the corresponding flowshop problem into an instance of the well-known asymmetric traveling salesman problem (ATSP) (e.g., [10]) and then finds a feasible schedule by solving this related ATSP. We then say that SETUP is a TSP-based heuristic.

3 Formulation of the Problem

Our flowshop environment consists of a set of n jobs that must be scheduled through a set of m machines. Each job has the same technological routing through machines. We consider that each job is available at time zero and has no due date. We also assume that there is a setup time which is sequence-dependent; that is, for every machine i there is a setup time before start processing a given task which depends on both the job to be processed (k) and the job that immediately preceeds this (j). We denote this setup time as s_{ijk}. Our objective is to minimize the time at which the last job in the sequence finishes processing on the last machine, also known as makespan. In [13], we provide a mixed integer programming (MIP) formulation of the problem. The MIP formulation can be useful when developing lower bounding schemes or attempting to find exact solutions by using some enumerative techniques such as branch-and-bound and cutting planes algorithms. For a discussion on how to apply these techniques, see Nemhauser and Wolsey [12].

4 Proposed Heuristics

4.1 The NEHT-RB() Heuristic

As pointed out in Section 2.1, the most effective heuristic heuristic for the general flowshop scheduling problem with makespan minimization (no setups) is due to Nawaz et al. [11], hereafter referred as the NEH heuristic (Nawaz, Enscore, and Ham). Taillard [17] develops a better way of implementing NEH. Here, we extend the NEH heuristic to handle setup times as well. We call this procedure NEHT-RB() (NEH modified by Taillard and extended by Ríos-Mercado and Bard).

The NEHT-RB() idea of building a feasible schedule is very simple. At each iteration of the algorithm there is a partial schedule S. A job h is selected from a list of unscheduled jobs P. The partial schedule S and the job h define a unique greedy function $\psi(j) : \{0, 1, \ldots, |S|\} \rightarrow R$, where $\psi(j)$ is the makespan of the new schedule S' resulting from inserting job h right after the j-th job in S. Here, position 0 means an insertion at the beginning of the schedule. Job h is inserted into position $k = \text{argmin}_j\{\psi(j)\}$; that is, the position in S that delivers the lowest makespan value of the partial schedule.

Figure 1 shows the pseudocode of NEHT-RB(). Its time complexity is $O(mn^2)$, where m is the number of machines and n is the number of jobs. A more detailed description of this heuristic is included in [13].

Procedure NEHT-RB()

Input: A set P of unscheduled jobs.

Output: A feasible schedule S.

Step 0. Set $S = \emptyset$

Step 1. Let P be the list of unscheduled jobs

Step 2. while $|P| > 0$ do

Step 2a. Remove h, the first job from P

Step 2b. Compute $\psi(j)$ for every position $j = 0, \ldots, |S|$

Step 2c. Choose $k = \text{argmin}_j\{\psi(j)\}$

Step 2d. Insert job h at position k in S

Step 3. Output S

Step 4. Stop

Figure 1: Pseudocode of NEHT-RB()

4.2 The GRASP Heuristic

A greedy randomized adaptive search procedure (GRASP), originally proposed by Feo and Resende [6], is a heuristic approach to combinatorial optimization problems that combines the power of greedy heuristics, randomization and local search techniques. GRASP consists of two phases: a construction phase and a postprocessing phase. During the construction phase, a feasible solution is built, one element at a time. At each iteration, the choice of the next element to be added is determined by ordering a subset of the remaining elements on a candidate list with respect to an appropiate greedy function. This function asseses the benefit of selecting each element. The probabilistic component of GRASP is implemented by choosing one element from the candidate list at random, not necessarily the top candidate. One way to limit the candidate list is by its cardinality where only the top λ elements are included.

The choice of the parameter λ requires insight into the problem. A compromise has to be made between being too restrictive or too loose. If the criterion used to form the list is too selective, only a few candidates will be available. The extreme case is when only one element is allowed. This corresponds to a pure greedy approach, and the same solution will be obtained every time GRASP is executed. The advantage of being restrictive in forming the candidate list

is that the greedy objective is not overly compromised. However, the danger of low variability is that the optimal and many very good solutions may not be encountered. GRASP phase 1 is applied N times, using different initial seed values each of which will generate a solution (in this case, a schedule) to the problem.

GRASP, originally introduced by Feo and Resende [6], has been applied succesfully to a number of combinatorial optimization problems. These include airline flight scheduling and maintenance base planning (Feo and Bard [4]), scheduling on parallel machines (Laguna and González-Velarde [8]), and railroad hitch assignment (Feo and González-Velarde [5]).

4.2.1 GRASP for the SDST Flowshop

Procedure GRASP()

Input: Set P of unscheduled jobs and λ, the size of the restricted candidate list.

Output: A feasible schedule S.

Step 0. Set $S = \emptyset$
Step 1. Let P be the set of unscheduled jobs
Step 2. while $|P| > 0$ do
Step 2a. Remove h, the first job from P
Step 2b. Compute $\psi(j)$ for every position $j = 0, \ldots, |S|$
Step 2c. Construct the RCL with the best λ positions
Step 2d. Choose randomly a position k from RCL
Step 2e. Insert job h at position k in S
Step 3. Output S
Step 4. Stop

Figure 2: Pseudocode of GRASP()

The GRASP construction phase is very similar to algorithm NEHT-RB() (see Section 4.1 for the notation). The difference between them is the way we choose the position in partial schedule S where the unscheduled job h is to be inserted.

In GRASP, the positions available for insertion are sorted by nondecreasing values of $\psi(j)$ and a restricted candidate list (RCL) is formed with the best λ positions. The probabilistic

strategy of GRASP selects one of the positions in the RCL randomly with equal probability. The job h is inserted at the selected position in S Figure 2 shows the pseudocode of the procedure (phase 1). GRASP phase 1 runs in $O(mn^2)$ time. A more detailed description is included in [13].

Notice that the extreme case of GRASP when $\lambda = 1$ corresponds to a pure greedy approach; that is, GRASP reduces to NEHT-RB().

5 Postprocessing Phase

In general a solution delivered by an approximation algorithm is not even a local minima. Thus, an attempt is made to improve upon the current feasible schedule S by examining its "neighborhood" (the set of schedules that are "close" to S) and making a move to a neighbor solution S' with a better value of the objective function (makespan, in this case). This new schedule S' defines a different neighborhood and the process iteratively repeats itself until no further improvement is possible, in which case we say that a local optimum (with respect to the given neighborhood) has been found. This iterative procedure is known as local search (sometimes also referred as postprocessing or improvement phase). The local search depends on how we define the neighborhood of a given feasible solution. In [13], we examine and computationally evaluate two different local search procedures. We observe that the procedure based on reinsertion of job strings proves superior. This procedure (S-job string reinsertion) is the one used in the present work.

6 Experimental Work

The purpose of the experiment is to evaluate the performance of each of heuristics including the postprocessing phase. To assess the quality of the solution delivered with respect to a lower bound (we do not know the optimal solution) we use a lower bounding scheme we developed in [13].

The GRASP() heuristic settings used are $\lambda = 2$ (which was found to be the best choice in a previous study [15]) and $N = 50$ iterations. The GRASP postprocessing phase is applied only once, to the best solution found in the construction phase. The procedures were coded in C++ and compiled with the Sun C++ compiler CC version 2.0.1 and optimization flag set to -O. CPU times are obtained by calling the clock() function in a SPARCStation 10.

To evaluate the procedures, 20 instances of the SDST flowshop are randomly generated for every combination $m \times n \in \{(2,4,6) \times (20,50,100)\}$ for two different classes of data sets: in

data set 1 (2) the setup times are uniformly distributed in the interval $[1, 99]$ ($[1, 10]$). In both data sets, the processing times are uniformly distributed in the interval $[1, 99]$.

For each combination of $m \times n$ we perform several comparions: (a) Summary statistics: we attempt to identify dominating characteristics by compiling objective function value and time related statistics such as number of times heuristic is best or tied for best, average percentage above lower bound, average CPU time, and worst CPU time; (b) Friedman test: this is a nonparametric test, analogue to the classical ANOVA test of homogenity, which we apply to the null hypothesis: all heuristic perform the same, against the alternate hypothesis: at least one of the heuristics do not perform the same as the others; and (c) Wilcoxon test: if Friedman test is significant; that is, the null hypothesis is rejected, we attempt to identify the "best" heuristic by performing a pairwise test among the heuristics. We apply the Wilcoxon signed rank test, a well-known nonparametric statistical test, to compare any two of the three heuristics. For more information on both Friedman and Wilcoxon tests, see Conover [1];

6.1 Experiment 1: Data Set 1 Instances

Summary of statistics										
		$n = 20$			$n = 50$			$n = 100$		
m	Statistic	N	S	G	N	S	G	N	S	G
2	Best	2	13	5	0	20	0	0	20	0
	Average % deviation	25	22	25	25	17	24	24	15	23
4	Best	6	6	8	2	17	1	0	20	0
	Average % deviation	43	44	42	43	39	43	44	35	44
6	Best	7	3	10	3	11	6	0	20	0
	Average % deviation	57	58	55	54	53	55	55	48	56

Table 1: Heuristic comparison for data set 1

Summary statistics on makespan value are shown in Table 1. Heuristics are identified by their initials (N, S, and G). For each cell, figures in first (Best) row indicate the number of times each heuristic found the best (or tied for best) solution. Figures in second row (Average % deviation) show the average percentage above lower bound. As we can see, SETUP() seems to dominate the other two for the larger instances (50- and 100-job data sets).

CPU time statistics are presented in Table 2. We observe that, on average, NEHT-RB() and SETUP() take the same amount of time, both of them being about three times faster that

CPU time (sec)										
		$n = 20$			$n = 50$			$n = 100$		
m	Statistic	N	S	G	N	S	G	N	S	G
2	Average	0.15	0.16	0.86	1.80	1.31	5.11	11.75	7.71	27.49
	Worst	0.26	0.24	0.96	4.26	1.98	6.76	18.50	11.37	37.44
4	Average	0.23	0.24	1.37	2.50	2.13	7.76	15.59	11.69	39.34
	Worst	0.44	0.37	1.55	5.94	3.46	9.63	36.70	20.18	51.34
6	Average	0.37	0.37	1.82	3.03	3.03	10.42	20.89	18.80	50.61
	Worst	1.15	0.69	1.96	5.48	5.18	12.38	46.58	28.98	63.60

Table 2: Summary of CPU time statistics for data set 1

GRASP(). It can also be learned from the table that SETUP() has a better worst-case time behavior than NEHT-RB().

Wilcoxon test results ($\alpha = 0.05$)			
m	$n = 20$	$n = 50$	$n = 100$
2	`setup()` best ($0.01 < p < 0.03$)	`setup()` best ($p < 0.02$)	`setup()` best ($p < 0.02$)
4	N/A	`setup()` best ($p < 0.02$)	`setup()` best ($p < 0.02$)
6	N/A	N/A	`setup()` best ($p < 0.02$)

Table 3: Wilcoxon test for data set 1

For those cells for which the Friedman test (not shown) was significant, a pairwise Wilcoxon test was performed between each pair of heuristics. These results are shown in Table 3. It is found that SETUP() outperforms the other two in all the 2-machine instances, and does even better as the size of input (number of jobs) grows. For the 4×20, 6×20, and 6×50 instances, the test was not significant implying that there is no difference among the heuristics.

6.2 Experiment 2: Data Set 2 Instances

Summary statistics on makespan value are shown in Table 4. Figures have the same meaning

Summary of statistics										
		$n = 20$			$n = 50$			$n = 100$		
m	Statistic	N	S	G	N	S	G	N	S	G
2	Best	9	2	13	4	9	8	3	15	3
	Average % deviation	2	3	2	1	1	1	1	1	1
4	Best	6	0	15	7	1	12	7	3	10
	Average % deviation	7	9	7	4	4	4	3	4	3
6	Best	9	1	10	9	1	11	8	2	10
	Average % deviation	14	18	14	7	8	7	5	6	5

Table 4: Heuristic comparison for data set 2

as those described in the previous section. We first should point out that the difference between the makespans delivered by the algorithms is very small. Although GRASP() seems to dominate in most of the instances, it does not beat NEHT-RB() by much. However, both procedures outperform SETUP(), with the exception of the 2-machine instances, where SETUP() does better as the number of jobs increases.

CPU time statistics for this data set 2 are very similar to those of data set 1. NEHT-RB() is, on average 30% to 80% faster than SETUP() and about 100% faster that GRASP(). Procedure NEHT-RB() also shows the minimum worst-case CPU time.

Wilcoxon test results ($\alpha = 0.05$)			
m	$n = 20$	$n = 50$	$n = 100$
2	$E[N] = E[G]$ NOT rejected ($p = 0.224$)	N/A	`setup()` best ($p = 0.003$)
4	`grasp()` better ($p = 0.022$)	$E[N] = E[G]$ NOT rejected ($p = 0.102$)	$E[N] = E[G]$ NOT rejected ($p = 0.274$)
6	$E[N] = E[G]$ NOT rejected ($p = 0.340$)	$E[N] = E[G]$ NOT rejeceted ($p = 0.176$)	$E[N] = E[G]$ NOT rejecetd ($p = 0.184$)

Table 5: Wilcoxon test for data set 2

For those cells where the Friedman test (not shown) was significant, a pairwise Wilcoxon test was performed. These results are shown in Table 5. As it can be seen, all the tests are significant at $\alpha = 0.05$, with the exception of the 2×50 set. Procedure SETUP() is found to be

statistically best for the 2×100 set, whereas GRASP() is observed to be better for the 4×20 set. All other tests show no significant difference between NEHT-RB() and GRASP(), which in turn outperform SETUP().

7 Conclusions

We have proposed two new (insertion-based) heuristics for the makespan minimization of the flowshop scheduling problem with setup costs. Both procedures were extensively evaluated and compared with the only previously existing heuristic (TSP-based) for this class of scheduling problem. The study also included a postprocessing phase based on job string reinsertion for finding local optimum.

Two different classes of randomly generated instances were considered, one where both processing and setup times are identically distributed (data set 1) and one where the setup times are small compared to the processing times (data set 2).

Our computational study revealed that

- Data set 1 instances are harder to solve in terms of relative gap performance. We observed that while our heuristics delivered near-optimal solutions for several of the data set 2 instances, the best solution (for data set 1) had, on the average, a relative gap of 15-22%, 35-42%, and 48-55% for the 2-, 4-, and 6-machine instances, respectively. However, further work remains to be done to determine the quality of the lower bounds for these type of instances.

- For data set 1, the TSP-based heuristic SETUP() was found to outperform the others with respect to both makespan (specially for the 50- and 100-job sets) and CPU time.

- For data set 2, the TSP-based heuristic worked better in the 2-machine instances; however, when the number of machine grows, the insertion-based heuristics NEHT-RB() and GRASP() dominated. This stems from the fact that the fewer the number of machines, the more the problem resembles an ATSP and should expect a TSP-based procedure to do well. The reader is reminded that in SETUP() the "distance" between jobs is computed as the sum of the setup times between jobs over all the machines. In the extreme case where there is only one machine, the problem reduces entirely to an instance of the ATSP. As we start considering more machines, this sum of setup times becomes less representative of the "distance" between the jobs. Between NEHT-RB() and GRASP() no statistical difference was found in terms of makespan value; however, the former runs considerably

faster.

This study provides practicioners a better way to find good schedules when facing flow line scenarios where setup times are small compared to the processing times. Procedures NEHT-RB() and GRASP() provided quick, near-optimal solutions for this class of instances. For researchers aiming at finding optimal solutions, the described procedures as well as the lower bounding scheme can be used to find quick upper and lower bounds that would enable the implementation of exact optimization algorithms.

8 Acknowledgments

The research of the first author was supported by the Mexican National Council of Science and Technology (CONACyT) and by The University of Texas at Austin. We also thank an anonymous referee whose suggestions helped improve the presentation of this paper.

References

[1] W. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, 1980.

[2] D. G. Dannenbring. An evaluation of flow shop sequencing heuristics. *Management Science*, 23(11):1174–1175, 1977.

[3] R. D. Dear. The dynamic scheduling of aircraft in the near terminal area. FTL Report R76-9, Massachussetts Institute of Technology, September 1976.

[4] T. A. Feo and J. F. Bard. Flight scheduling and maintenance base planning. *Management Science*, 35(12):1415–1432, 1989.

[5] T. A. Feo and J. L. González-Velarde. The intermodal assignment problem: Models, algorithms, and heuristics. Technical Report ORP90-10, Operations Research Program, University of Texas at Austin, August 1990.

[6] T. A. Feo and M. G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, 1989.

[7] C. A. Glass and C. N. Potts. A comparison of local search methods for permutation flow shop scheduling. *Annals of Operational Research*, 1995. (To appear).

[8] M. Laguna and J. L. González-Velarde. A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, 2:253–260, 1991.

[9] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S. S. Graves, A. H. G. Rinnooy Kan, and P. Zipkin, editors, *Handbook in Operations Research and Management Science, Vol. 4: Logistics of Production and Inventory*, pages 445–522. North-Holland, New York, 1993.

[10] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons, Chichester, 1990.

[11] M. Nawaz, E. E. Enscore Jr., and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA The International Journal of Management Science*, 11(1):91–95, 1983.

[12] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1988.

[13] R. Z. Ríos-Mercado and J. F. Bard. The flowshop problem with setup costs: Lower bounds, heuristics, and local search. Working Paper, University of Texas at Austin, September 1995.

[14] R. Z. Ríos-Mercado and J. F. Bard. The flowshop scheduling problem: A survey. Working Paper, Operations Research Program, University of Texas at Austin, August 1995.

[15] R. Z. Ríos-Mercado and J. F. Bard. The flowshop scheduling problem with sequence-dependent setup times: A GRASP approach. Working Paper, Operations Research Program, University of Texas at Austin, July 1995.

[16] J. V. Simons Jr. Heuristics in flow shop scheduling with sequence dependent setup times. *OMEGA The International Journal of Management Science*, 20(2):215–225, 1992.

[17] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74, 1990.

[18] S. Turner and D. Booth. Comparison of heuristics for flow shop sequencing. *OMEGA The International Journal of Management Science*, 15(1):75–85, 1987.

[19] M. Widmer and A. Hertz. A new heuristic method for the flow shop sequencing problem. *European Journal of Operational Research*, 41(2):186–193, 1989.