



## Heuristic-Programming Solution of a Flowshop-Scheduling Problem

Martin J. Krone; Kenneth Steiglitz

*Operations Research*, Vol. 22, No. 3. (May - Jun., 1974), pp. 629-638.

Stable URL:

<http://links.jstor.org/sici?sici=0030-364X%28197405%2F06%2922%3A3%3C629%3AHSOAFP%3E2.0.CO%3B2-6>

*Operations Research* is currently published by INFORMS.

---

Your use of the JSTOR archive indicates your acceptance of JSTOR's Terms and Conditions of Use, available at <http://www.jstor.org/about/terms.html>. JSTOR's Terms and Conditions of Use provides, in part, that unless you have obtained prior permission, you may not download an entire issue of a journal or multiple copies of articles, and you may use content in the JSTOR archive only for your personal, non-commercial use.

Please contact the publisher regarding any further use of this work. Publisher contact information may be obtained at <http://www.jstor.org/journals/informs.html>.

Each copy of any part of a JSTOR transmission must contain the same copyright notice that appears on the screen or printed page of such transmission.

---

The JSTOR Archive is a trusted digital repository providing for long-term preservation and access to leading academic journals and scholarly literature from around the world. The Archive is supported by libraries, scholarly societies, publishers, and foundations. It is an initiative of JSTOR, a not-for-profit organization with a mission to help the scholarly community take advantage of advances in technology. For more information regarding JSTOR, please contact [support@jstor.org](mailto:support@jstor.org).

# Heuristic-Programming Solution of a Flowshop-Scheduling Problem

Martin J. Krone

*Bell Telephone Laboratories, Whippany, New Jersey*

and

Kenneth Steiglitz

*Princeton University, Princeton, New Jersey*

(Received April 5, 1971)

This paper considers the static flowshop-scheduling problem with the objective of minimizing, as a cost function, the mean job-completion time. Within the more general framework of combinatorial optimization problems, it defines a heuristic search technique—an approach that has been successful in the past in obtaining near-optimal solutions for problems that could not be solved exactly, either for lack of theory or because of exorbitant computational requirements. The paper presents a two-phase algorithm: The first phase searches among schedules with identical processing orders on all machines; the second refines the schedule by allowing passing. Results of computer study are presented for a large ensemble of pseudorandom problems, and for two particular problems previously cited in the literature. The method is shown to provide solutions that are exceptionally low in cost, and superior to those provided by sampling techniques in the cases for which comparison is possible. Computation time is also discussed and is given in machine-independent terms.

**T**HE PURPOSE of this paper is to present a heuristic method used for obtaining near-optimal solutions to large static flowshop-scheduling problems with a mean-flow-time cost criterion. The method of approach, referred to here as *heuristic programming*, is applicable to combinatorial optimization problems in general, and was developed largely in the context of the travelling salesman problem by REITER AND SHERMAN<sup>[8]</sup> and LIN<sup>[9]</sup>. More recently, the method has been applied with some success to several other problem areas (for example, *see* references 2, 5, and 9).

The assumptions and requirements of the static flowshop-scheduling problem considered here are:

1. The machine group consists of  $m$  machines  $M_1, M_2, \dots, M_m$ , each performing a different function.
2. All machines are available and ready to start processing at  $t=0$ .
3. There are  $n$  independent jobs to be scheduled, identified by integers  $1, 2, \dots, n$ .
4. Each job consists of  $m$  operations whose precedence structure is a strict ordering of the operations, where, for each job, the first operation requires machine  $M_1$ , the second operation requires  $M_2$ , etc.

5. The required processing times (denoted  $t[i, j]$  for the  $i$ th operation of the  $j$ th job) are known in advance.

6. No interruption of an operation is allowed—each must be scheduled into a single contiguous time interval.

7. The cost of a schedule  $S$ , denoted by  $C(S)$ , is the flow time averaged over the jobs, where flow time for a job is the time elapsed from  $t=0$  until the completion of its  $m$ th operation (the earliest time at which it may leave the system).

A review of the flowshop scheduling problem can be found in CONWAY, MAXWELL, AND MILLER.<sup>[1]</sup> Following their notation, an  $n$ -job,  $m$ -machine flowshop problem is abbreviated as an  $n/m$  problem.

The constraints of the flowshop problem do not dictate the order in which the jobs are taken by a particular machine and, in fact, this order may differ from one machine to another. If the same order is used by every machine, the resulting schedule is called *order preserving*.

## 1. SAMPLING APPROACHES AND THEIR INFERENCES

THE INVESTIGATION OF HELLER<sup>[2]</sup> was one of the earliest attempts to understand the contributing factors to the cost of large flowshop schedules, and to evaluate the utility of a sampling technique. To this end, Heller computed histograms of finishing time for randomly sampled schedules for a particular 20/10 problem, comparing the distribution of finishing time for order-preserving schedules with that of the more general class, in which the order for each machine was generated independently. Finding the worst order-preserving schedule obtained far better than the best general schedule obtained, he concluded that the former represents a more attractive investment of computation although possibly excluding optimal schedules. Because the general schedules tend to include exorbitant delays, the conclusion carries over to mean flow time.

The work of NUGENT<sup>[7]</sup> on schedule sampling by randomized dispatching considers the flowshop problem as a special case of the more general jobshop problem. His work presents results on sample problems whose data is available in the open literature; some of these sample problems are also used here. In connection with flowshop schedules, Nugent examined the structure of the best schedules and, based on these results, made several noteworthy observations concerning controlled departure from order-preservation that appear likely to improve upon schedules. On his schedules for the same 20/10 problem studied by Heller, he observes that they "did not strictly preserve order over the 10 machines, but rather specified occasional and local reversals of order on neighboring machines." In other words, jobs should be allowed to 'pass' each other as they move from one machine to the next.

## 2. COMBINATORIAL OPTIMIZATION PROBLEMS AND HEURISTIC PROGRAMMING

INTUITIVELY, A COMBINATORIAL optimization problem is of the following nature: given a structure that can be arranged in any one of a large but finite number of

ways, and some numerical data that allow a 'cost' to be associated with each, design the structure so as to minimize the cost. For the forthcoming discussion of a heuristic program, the notion of such a problem is abstracted as below.

A *combinatorial optimization problem* is a triple  $(S, X, f)$  with the following connotation:

(a) The *solution space*  $S = \{s_1, s_2, \dots, s_N\}$  is a finite set of *feasible solutions* that satisfy structural requirements of the problem and are thus candidates for optimality.

(b) The *parameter space*  $X$ ; each point  $x \in X$  represents admissible 'data' for the problem.

(c) The *cost function*  $f: S \times X \rightarrow R$  where  $f(s, x)$  is the cost of solution  $s$  with parameter value  $x$  ( $R$  is the set of real numbers).

A *globally optimal solution* (or simply an optimal solution) for parameter value  $x$  is a feasible solution  $s^*$  such that  $f(s^*, x) \leq f(s, x)$  for all  $s \in S$ .

In the sequel,  $x$  is assumed fixed through the optimization process; hence  $f(s, x)$  will be denoted simply by  $C(s)$ .

Based on the apparent structure of solutions, a 'perturbation' rule, called a *local transformation*, is designed, which allows a given solution to be transformed into any one of a set of alternative solutions. These alternatives may be thought of as neighboring the given solution in solution space. (Hence the use of the word 'local.' A local transformation need not be localized within the solution structure—the notion of locality applies in solution space only.) Given this structure, the idea of local optimality follows naturally.

A *neighborhood structure* on a solution space  $S$  is an undirected graph  $N$  on  $S$ .

For each  $s \in S$ , the *neighborhood* of  $s$ , denoted  $N(s)$ , consists of  $s$  and all points adjacent to  $s$  in  $N$ .

A solution  $s$  is *locally optimal* in a neighborhood structure if  $C(s) \leq C(s')$  for every  $s'$  in  $N(s)$ .

Locally optimal solutions can be exceptionally low in cost (and have a reasonable probability of being optimal) if the local transformation is well chosen. The goal of a heuristic program is to compute members of the set of locally optimal solutions as now described. To initiate the computation, a starting solution is chosen pseudorandomly.

A *pseudorandom starting algorithm* is an algorithm that, for each call, generates a member of  $S$  so as to simulate some probability distribution over  $S$ .

A *heuristic program* is an algorithm that, using a pseudorandom starting algorithm and a local transformation  $N$ , computes a sequence  $s_1, s_2, \dots, U(s_1)$  of trial solutions such that:

- (1)  $s_1$  is generated by the pseudorandom starting algorithm.
- (2)  $s_i \in N(s_{i-1})$  and  $C(s_i) < C(s_{i-1})$ ,  $i = 2, 3, \dots$ .
- (3)  $U(s_1)$ , the 'ultimate successor'<sup>[8]</sup> of  $s_1$ , is locally optimal in  $N$ .

The program terminates when local optimality of the current trial solution is verified by searching its neighborhood exhaustively. In application to the flowshop problem, incumbent solutions are replaced on a first-improvement-found basis. For each run of the program from a new starting solution  $s_1$ , a sample of the local optima is computed. The program is usually used by making as many runs as resources allow, and taking the best of the results.

### 3. APPLICATION TO THE FLOWSHOP PROBLEM

#### 3.1 Notation and Formalism

Following conventional terminology, a schedule is called *semiactive* if no operation in it can be continuously shifted forward in time, i.e., the schedule is 'compact.' The flowshop problem is reduced to a combinatorial one by considering as feasible solutions only semiactive schedules. Since all schedules are semiactive, a unique schedule with associated cost is implied given the processing times and the job processing order for each machine. (An algorithm for computing cost is given in the next section.) With this assumption, the following identification is adopted: *A schedule for an  $n$ -job,  $m$ -machine flowshop problem is an  $m \times n$  matrix  $S$  in which each row contains each of the integers  $1, 2, \dots, n$  exactly once.*

The connotation of  $S$  is that  $S_{i,j}$  is the job number of the job that is  $j$ th to be processed on machine  $M_i$  under schedule  $S$ . Note that, for fixed  $i$ ,  $S$  defines (as a function of  $j$ ) a permutation of degree  $n$  that is the processing order for the jobs on machine  $M_i$ . This permutation is described by the  $i$ th row vector of  $S$  and is denoted by  $S_{i,\bullet}$ .

In all other cases, the double subscript  $i,j$  refers directly to the  $i$ th operation of job number  $j$ . The following quantities are relevant:

$t[i, j]$  = operation processing time.

$s[i, j]$  = operation starting time in schedule.

$f[i, j]$  = operation completion time in schedule.

The start and finishing times are those in the semiactive schedule represented by  $S$ . The cost (mean flow time) may then be expressed as

$$C(S) = (1/n) \sum_{j=1}^{j=n} f[m, j].$$

As examples of the notation, and quantities to be used below,  $t[i, S_{i,j}]$  is the processing time on the  $i$ th machine of the job scheduled  $j$ th in order on that machine;  $f[i-1, S_{i,j}]$  is the earliest time at which job  $S_{i,j}$  is ready for processing of its  $i$ th operation ( $i > 1$ );  $f[i, S_{i,j-1}]$  is the time at which  $M_i$  is free to take on job  $S_{i,j}$  ( $j > 1$ ); a schedule is order preserving if  $S_{i,k} = S_{j,k}$  for all  $i, j$ , and  $k$  or, equivalently,  $S_{i,\bullet} = S_{j,\bullet}$  for all  $i$  and  $j$ .

#### 3.2. Schedule-Cost Evaluation

In order to compute the cost of a schedule, it is necessary to form the schedule explicitly, assigning starting and finishing times to each operation. An algorithm for computing the cost  $C(S)$  of the (semiactive) schedule corresponding to  $S$  works in the following manner:

1. Schedule job operations on the first machine in order given by  $S_{1,\bullet}$  with no idle time between operations.

2. (a) Perform Step 2(b) for  $i = 2, 3, \dots, m$ .

- (b) Schedule jobs on machine  $M_i$  in the order given by  $S_{i,\bullet}$ , starting job  $S_{i,j}$  at the earliest feasible time, given by

$$s[i, S_{i,j}] = \begin{cases} \max \{f[i-1, S_{i,j}], f[i, S_{i,j-1}]\}, & (j > 1) \\ f[i-1, S_{i,j}], & (j = 1) \end{cases}$$

and set  $f[i, j] = s[i, j] + t[i, j]$ .

The algorithm consists of  $mn$  basic steps; each involves taking the maximum of two quantities already computed and adding an operation time to the result. Computation therefore increases in proportion to the size of the problem, but with a small constant of proportionality.

### 3.3. Design of the Heuristic Program

The program for flowshop scheduling is based primarily on the results of preliminary computational experiments that were concerned with observing the performance of a fairly general class of transformations on schedules in order to surmise transformations capable of finding improvements efficiently. Details of these experiments, together with more extensive treatment of topics discussed in this section, are given by Krone.<sup>[6]</sup>

Results of this investigation agreed strongly with conclusions drawn from sampling studies discussed in Section 1: that the set of order-preserving schedules should be privileged, but that consideration should be given to limited deviation from order-preservation with the possibility of further improvement. They also suggested a transformation rule that would fill the role of Nugent's "occasional and local reversals"; this appears in phase 2 of the optimization process discussed below.

The heuristic program computes locally optimal schedules in two phases. In phase 1, search is confined to the set of order-preserving schedules; in phase 2, the schedule produced by phase 1 is subject to deviation from uniform ordering. The two phases are independent, so that phase 2 can start from any order-preserving schedule, and phase 1 can use any local transformation defined on permutations.

#### Phase 1 Optimization

To start phase 1,  $S$  is defined pseudorandomly, with a uniform distribution over the set of order-preserving schedules, by the following algorithm.

1. Generate a random permutation  $p$  of degree  $n$ .
2. (a) Perform step 2(b) for  $i = 1, 2, \dots, m$ .  
 (b) Set  $S_{i,j} = p(j)$  for  $j = 1, 2, \dots, n$ .
3. Compute  $C(S)$  using the cost-evaluation algorithm.

Several transformation rules were suggested by the experimental results. The rule selected for further study, referred to as *single insertion*, may be stated thus: for any  $i$  and  $j$  such that  $1 \leq i < j \leq n$ , remove the  $j$ th job in the sequence and reinsert it in the  $i$ th position. For example, the transformation

1 2 3 **4** 5 6 7 8 9 — 1 2 3 7 4 5 6 8 9

that moves the bold-face job to the space marked, corresponds to  $i = 4, j = 7$  (i.e., the seventh job has been moved to the fourth position).

It has also been discovered that the number of calls to the cost-evaluation algorithm, and therefore the computation time, during phase 1 can be quite sensitive to the order in which neighborhoods are searched. This is determined by the exact manner in which indexing is done in program implementation. Letting ordered pairs  $(i, j)$  denote single insertions as suggested above (where  $1 \leq i < j \leq n$ ), the ordering imposed upon transformations by program indexing is

$$(i, j) < (k, l) \text{ iff } i < k, \text{ or } i = k \text{ and } j < l.$$

When an improvement is found, the corresponding values of  $i$  and  $j$  are recorded, and indexing continues from the next value  $(i, j+1)$ , reverting to  $(1, 2)$  whenever the highest value  $(n-1, n)$  is reached. A permutation is therefore locally optimal upon rearrival at the point of last improvement.

### Phase 2 Optimization

The second phase begins with the order-preserving schedule produced in phase 1, and attempts to reduce  $C(S)$  further by allowing deviations from uniform ordering.

The local transformation used in phase 2 consists of rearrangements of  $S$  of the following type: for any  $j$ ,  $1 \leq j \leq n-1$ , and  $k$ ,  $2 \leq k \leq m$ , interchange  $S_{i,j}$  with  $S_{i,j+1}$  for  $i = k, k+1, \dots, m$ . The value of  $k$  is strictly nondecreasing through phase 2, with all such interchanges considered for each value of  $k$  before proceeding to the next value. In effect, a different 'subneighborhood' is defined for each value of  $k$ , and checkout of each is completed before moving to the next. In the following algorithm, step 2 searches a subneighborhood and step 3 moves to the next.

1. Set  $k=2$ .
2. (a) Set  $j=1$ .
  - (b) If interchange of  $S_{i,j}$  with  $S_{i,j+1}$  for  $i = k, k+1, \dots, m$  yields a schedule of lower cost, accept the change and return to part (a). If not, continue.
  - (c) If  $j < n-1$ , increase  $j$  by 1 and repeat part (b); if  $j = n-1$ , go to step 3.
3. If  $k < m$ , increase  $k$  by 1 and repeat step 2; if  $k = m$ ,  $S$  is phase 2 optimal. Stop.

Thus, for example, starting from the natural-order-preserving schedule for an 8/5 problem, phase 2 might produce the schedule

$$\begin{array}{cccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
 S= & 1 & 2 & 3 & 4 & 5 & 7 & 6 & 8 \\
 & 1 & 2 & 4 & 3 & 5 & 7 & 6 & 8 \\
 & 1 & 2 & 4 & 3 & 5 & 7 & 8 & 6
 \end{array}$$

by the following transformations: interchange of positions 6 and 7 for the last three machines, then positions 3 and 4 for the last two machines, then positions 7 and 8 for the last machine. These transformations correspond to the 'passing' or 'local reversals' discussed earlier.

## 4. COMPUTATIONAL RESULTS

## 4.1. Locally Optimal Schedules for Pseudorandom Problems

For the statistical studies cited here, the algorithms described above were implemented in FORTRAN IV. An ensemble of 50 pseudorandom problems of size 10/5 was developed, this size being large enough for the results to be typical of the heuristic program, yet small enough that large numbers of runs could be made economically. Processing times were uniformly distributed on the interval (0, 1000) and truncated to integer values so that integer mode computation could be used. The choice of the uniform distribution is in keeping with established practice in the literature on the flowshop problem.

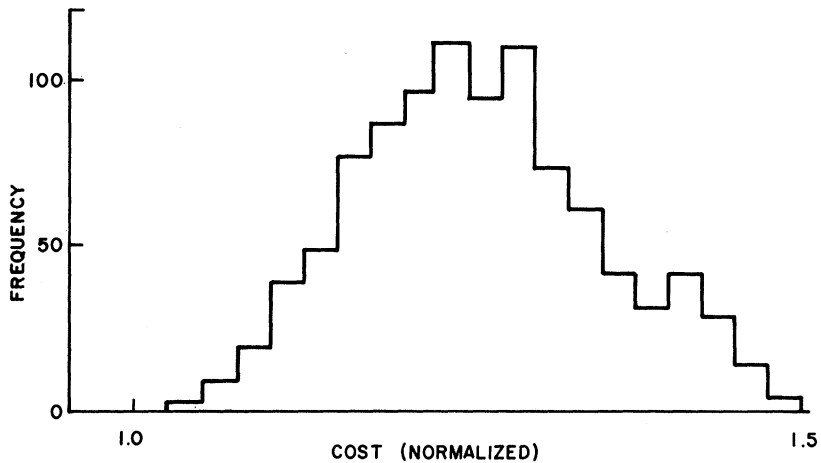


Fig. 1. Cost distribution for pseudorandom order-preserving starting schedules.

For the purpose of standardizing the results, an empirical optimal cost was established for each of the 50 problems in the ensemble, as follows. For each problem, twenty runs of the heuristic program were made; the best of the 20 phase 2 locally optimal schedule costs was taken as the empirical optimum and all 20 samples were normalized to it. The costs of the corresponding pseudorandom starting schedules were similarly normalized to this empirical optimum and saved; these reflect a random sampling of order-preserving schedules.

Figures 1 and 2 present histograms of the 1000 normalized costs computed as above; starting values are included in the histogram of Fig. 1 and phase 2 locally optimal costs in that of Fig. 2. Comparing the cost distribution of locally optimal schedules with that of the general population of order-preserving schedules, as reflected in the starting costs, we can see that virtually all the locally optimal costs fall within the lower 0.3 percentile of the general population distribution.

In 41 of the 50 problems, some improvement occurred in phase 2 of one or more

of the 20 runs; for the runs in which phase 2 produced some improvement, the average decrease in cost was about 0.66 percent of the cost at the end of phase 1. The average number of cost evaluations for these runs was approximately 128 for phase 1 and 38 for phase 2.

Table I summarizes results set forth in this section and the next, and includes some additional information.

#### 4.2. Performance on Previously Explored Problems

The performance of the heuristic program has been tested on two sample problems for which the processing times appear in published work: the 20/10

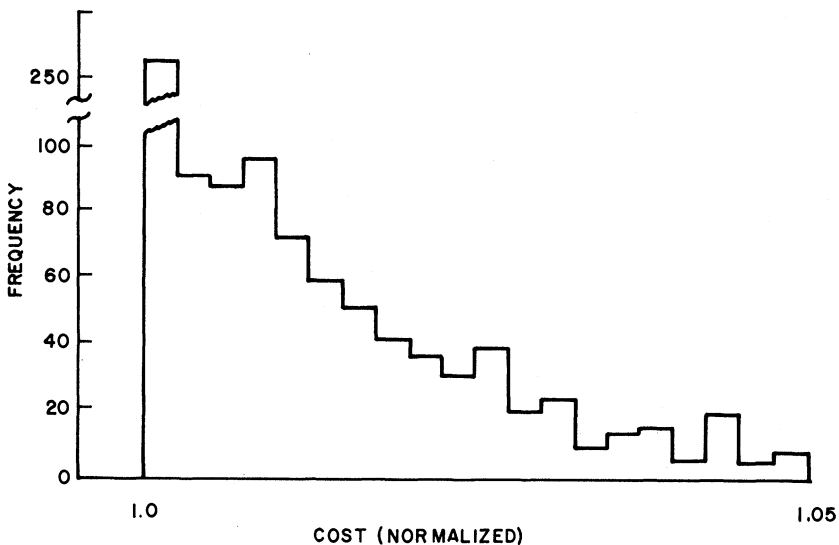


Fig. 2. Cost distribution for phase 2 locally optimal schedules.

problem used by Heller in his random-sampling study (the first 20 of 100 jobs given in reference 3, Table I), and a 9/3 problem used by STORY AND WAGNER in studying integer programming (reference 10, Table 5-D). The figures given by Nugent<sup>[7]</sup> for the mean flow times of sample dispatching schedules for these problems provide measuring marks.

For the 9/3 problem, the best of 1300 samples of nondelay schedules formed by Nugent's probabilistic dispatching procedure had a mean flow of 45.78. Using a modified procedure that allowed controlled delay insertion, the best of 1000 further samples had a cost of 44.11. For the same problem, the heuristic program was run 100 times; the best schedule produced, with a cost of 43.33, appeared 70 times. (The second-best schedule, with a cost of 44.66, appeared 29 times, and one schedule had a cost of 45.0.) No improvement was produced by phase 2 on any of the 100 runs. The average number of cost evaluations was approximately 109 in phase 1 and 16 in phase 2 (the figure for phase 2 was always 16, since this phase amounts to a checkout of all possible reversals with none actually done).

For the 20/10 problem, the best nondelay schedule of 1300 samples generated by Nugent had a cost of 98.75; further sampling of 100 delay schedules reduced the cost to 92.75. In 20 runs of the heuristic program, the best phase 1 optimal schedule had a cost of 88.8; the best phase 2 optimal schedule had a cost of 88.6 and resulted from a phase 1 optimal schedule whose cost was 89.45. The average of the non-zero cost reductions in phase 2, as a percentage of the cost at the end of phase 1, was about 0.3 percent. Phase 1 required an average of 694 cost evaluations, and phase 2 an average of 183.

## 5. CONCLUSIONS

PREVIOUS EXPERIENCE WITH the flowshop problem, together with experimental study of local transformations, aided the design of an efficient heuristic program.

TABLE I  
SUMMARY OF COMPUTATIONAL RESULTS

Problem Description	Source of data	Best mean flow time by randomized dispatching (Nugent [7])		Heuristic program results					
				Final schedule cost				Avg. no. of cost evaluations	
				Phase 1		Phase 2			
				Non-delay	Delay	Best	Avg.	Best	Avg. improvement when nonzero
Ensemble of 50 10/5 problems 9/3  20/10	Pseudorandom	—	—	—	—	—	0.66%	128	38
	Ref. 10, Table 5-D	45.78	44.11	43.33	43.66	43.33	None observed	108	16
	Ref. 3 Table I	98.75	92.75	88.8	90.0	88.6	0.3%	694	183

Execution time of the program is sensitive to the exact manner in which the neighborhood search is indexed. Almost all of the minimization of mean flow time is accomplished in phase 1, with phase 2 of the optimization process providing only slight improvement in general. However, for certain instances of processing times, the cost reduction in phase 2 can be appreciable, and this possibility should not be overlooked.

## ACKNOWLEDGMENTS

THIS WORK STEMS from a Princeton University Ph.D. thesis that was supported in part by the U.S. Army Research Office—Durham, and by the Air Force Office of Scientific Research. The computer facilities used were supported in part by the National Science Foundation.

## REFERENCES

1. RICHARD W. CONWAY, W. L. MAXWELL, AND L. W. MILLER, *Theory of Scheduling*, Addison-Wesley, Reading, Mass., 1967.
2. F. J. GRATZER, "Computer Solutions of Large Multicommodity Flow Problems," Ph.D. thesis, Princeton University, 1970.
3. J. HELLER, "Some Numerical Experiments for an  $M \times J$  Flow Shop and its Decision-Theoretic Aspects," *Opns. Res.* **8**, 178-184 (1960).
4. I. N. HERSTEIN, *Topics in Algebra*, Blaisdell, Waltham, Mass., 1964.
5. M. J. KRONE, "Heuristic Programming Applied to Scheduling Problems," Ph.D. thesis, Princeton University, 1970.
6. S. LIN, "Computer Solutions of the Traveling Salesman Problem," *Bell System Tech. J.* **44**, 2245-2269 (1965).
7. C. E. NUGENT, "On Sampling Approaches to the Solution of the  $N$ -by- $M$  Static Sequencing Problem," Ph.D. thesis, Cornell University, 1964.
8. S. REITER AND G. SHERMAN, "Discrete Optimizing," *SIAM J.* **13**, 864-889 (1965).
9. K. STEIGLITZ, P. WEINER, AND D. KLEITMAN, "The Design of Minimum Cost Survivable Networks," *IEEE Trans. Circuit Theory* **CT-16**, 455-460 (1969).
10. A. E. STORY AND H. M. WAGNER, "Computational Experience with Integer Programming for Job-Shop Scheduling," Chap. 14 in J. F. MUTH AND G. L. THOMPSON (eds.), *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, N. J., 1963.