MANAGEMENT SCIENCE
Vol. 38, No. 10, October 1992
Printed in U.S.A.

NEW SEARCH SPACES FOR SEQUENCING PROBLEMS WITH APPLICATION TO JOB SHOP SCHEDULING*

ROBERT H. STORER, S. DAVID WU AND RENZO VACCARI
Department of Industrial Engineering, Lehigh University, Bethlehem, Pennsylvania 18015

In this paper search heuristics are developed for generic sequencing problems with emphasis on job shop scheduling. The proposed methods integrate problem specific heuristics common to Operations Research and local search approaches from Artificial Intelligence in order to obtain desirable properties from both. The applicability of local search to a wide range of problems, and the incorporation of problem-specific information are both properties of the proposed algorithms. Two methods are proposed, both of which are based on novel definitions of solution spaces and of neighborhoods in these spaces. Applications of the proposed methodology are developed for job shop scheduling problems, and can be easily applied with any scheduling objective. To demonstrate effectiveness, the method is tested on the job shop scheduling problem with the minimum makespan objective. Encouraging results are obtained.
(LOCAL SEARCH; SEARCH NEIGHBORHOODS; SCHEDULING; COMBINATORIAL OPTIMIZATION)

1. Introduction

Combinatorial optimization problems form a diverse set which include many scheduling problems and are characterized by an inherent difficulty of solution. Many such problems, including job shop scheduling (JSP), are NP-hard indicating the intractability of large and often moderately sized problems. Optimal solution procedures typically rely on implicit enumeration methods such as branch and bound. While much progress has been made in this approach, job shop scheduling has proven to be particularly difficult. As a result of this inherent intractability, heuristic solution procedures are an attractive alternative.

In this paper we present methods by which effective local search heuristics can be simply developed for sequencing and scheduling problems. The methods are illustrated by applying them to the job shop scheduling problem. An interesting feature of the heuristics is that they simultaneously employ problem-specific heuristics familiar to readers of the operations research literature, and local search methods more frequently associated with the artificial intelligence field. The key to local search is the definition of a solution neighborhood, a problem that has been termed "an art" which is "usually guided by intuition" (Papadimitriou and Steiglitz 1982). Most current implementations rely on naive search neighborhoods which fail to exploit problem specific knowledge. It could be said that these methods address how to search the solution space, not where to search.

The basis for the proposed methods is to integrate fast, problem-specific (typically greedy) heuristics with local search. The key concept is to base the definition of a search neighborhood on a heuristic, problem pair (h, p) where h is a fast, known, problem-specific heuristic and p represents the problem data. Since a heuristic h is a mapping from a problem to a solution, the pair (h, p) is an encoding of a specific solution. By "perturbing" the heuristic h, the problem p, or both, a subset, or neighborhood of solutions, is generated. This neighborhood forms the basis for local search.

Due to their generic nature, the proposed methods possess several desirable properties. First, the basic principle can be applied to develop heuristics for a broad spectrum of problems. Second, because emphasis is placed on developing search spaces, various search

* Accepted by Thomas M. Liebling; received April 1990. This paper has been with the authors 4 months for 1 revision.

0025-1909/92/3810/1495$01.25

strategies such as simulated annealing, genetic algorithms, or Tabu search can be applied. Third, as applied to scheduling, it is objective independent. That is, it can be applied to scheduling problems with various constraints and/or any regular or nonregular objective.

2. Background

Heuristic solution procedures for combinatorial optimization in general, and the job shop scheduling problem (JSP) in particular, have been widely studied. Most heuristics fall in the "problem specific" class common to Operations Research. These heuristics are developed for each problem type and rely on problem-specific knowledge. Of particular relevance to this research are fast heuristics, typically of the greedy variety. These heuristics form the basis for the search space and neighborhood definitions (they are the heuristic in the heuristic-problem pair). A second class of heuristics are local search methods, examples of which include hill climbing, steepest descent, simulated annealing, Tabu search, and Genetic Algorithms. These methods provide alternative ways to search solution space based on the proposed neighborhood definitions. In the remainder of this section a brief summary of pertinent problem-specific heuristics is presented, followed by a survey of local and probabilistic search methods that have been applied to scheduling problems.

2.1. *Problem-Specific Heuristics*

Problem-specific heuristics for the JSP are usually based on a "divide and conquer" approach. Greedy heuristics are the simplest form of such methods in which the problem is partitioned into a sequence of simpler problems. Due to considerations regarding computational speed, "fast" heuristics are of primary relevance to this paper. In particular, well-known heuristics based on priority dispatching rules will be relied on heavily. Examples of commonly used dispatching rules include shortest processing time first (SPT), most work remaining (MWR), and earliest due date (EDD). Panwalker and Iskander (1977) list over 100 such rules. Extensions of these methods based on linear combinations of dispatching rules (cf. Panwalkar and Iskander 1977) and time sequenced lists of dispatching rules (Wu 1987) have also been studied extensively. Other dispatching heuristics based on Gittins indices have also been proposed (Gittins 1979). Dispatching rules will prove useful for "heuristic based" solution encodings, and are discussed further in §4.

An additional important heuristic is the "Shifting Bottleneck" heuristic of Adams et al. (1988) for the JSP with makespan objective. This heuristic has been shown to achieve excellent results, and will serve as a basis for comparison in §5.

2.2. *Local Search Heuristics for Sequencing Problems*

A resurgence of interest in local search has occurred in recent years due to the development of probabilistic local search methods such as simulated annealing (SA), genetic algorithms (GA), and Tabu search (TS). These methods have been frequently applied to sequencing problems, most often the traveling salesman problem (TSP). SA applications to the TSP may be found in Bonomi and Lutton (1984), Cerny (1985), Golden and Skiscim (1986), Kirkpatrick et al. (1983), Kirkpatrick (1984), and Lundy and Mees (1986). Of note is the fact that all of these applications are based on 2-opt (swap) neighborhoods. The results of these endeavors have been mixed. In its current form SA seems to be most promising for very large problems. Golden and Skiscim (1986) give empirical evidence that SA is generally inferior to known problem-specific and composite heuristics for TSP's in the range of 50 to 100 cities. Few if any applications of SA to scheduling problems have been attempted.

Genetic algorithms have also been frequently applied to the TSP with limited success. Examples include Goldberg and Lingle (1985), Oliver et al. (1987), Greffenstette et al. (1985), Greffenstette (1987), Suh and Van Gucht (1987), and Liepins et al. (1987).

Davis (1985) proposed a novel problem representation to develop a GA for a simplified JSP with two job types and set up costs. The method utilizes job preference lists constructed for each machine in each of several time windows. The list consists of job types and the actions "wait" and "idle". A decoding procedure constructs a schedule from the preference lists in order to evaluate the objective function. More recently, Whitley et al. (1989) describe an edge based recombination operator which is used to develop a GA for job shop scheduling.

Tabu search (Glover 1986, 1989, 1990) is another local search method that has been applied to sequencing problems with promising results. Applications include the TSP (Glover 1989), the JSP (Taillard 1989; Eck and Pinedo 1989), and one-machine scheduling with set up costs and delay penalties (Laguna et al. 1989).

3. New Neighborhoods for Local Search

Current applications of local and probabilistic search methods to sequencing problems are based on neighborhoods defined in the solution space of the problem. Typically this definition is based on "swapping" operators. Examples include swapping arcs in a traveling salesman tour, swapping nodes between sets in graph partitioning, or swapping jobs in the sequence of a scheduling problem. While local and probabilistic search based on these neighborhoods has been described as promising (CONDOR 1988), results to date are not competitive with standard problem-specific heuristics. A hypothesized explanation is that simple neighborhoods based on swapping are naive in that they fail to exploit problem-specific information. Conversely, problem-specific heuristics are specially designed to exploit problem structure, and thus have an inherent advantage over naive local search methods. As an alternative we propose local search methods based on search spaces and neighborhood definitions which explicitly incorporate known problem-specific heuristics as a means to exploit problem structure. The proposed neighborhood definition is based on the fact that a heuristic is a mapping from a problem to a solution; $h(p) = s$. Thus a heuristic, problem pair (h, p) is an encoding of a solution. A subset of the solution space can be generated by a set of heuristics:

$$H = \{h_i, i = 1, \ldots, n\}.$$

That is, the n heuristics generate a subset of solutions:

$$S = \{h_i(p), i = 1, \ldots, n\}.$$

Similarly, a subset of solutions may be generated by the application of a single heuristic to perturbed versions of the original problem. Let P be a set of m problems generated by perturbing the original problem data (e.g., by perturbations to processing times in a JSP, city coordinates in a TSP, etc.). That is:

$$P = \{p_j = p_0 + d_j, j = 1, \ldots, m\}$$

where p_0 is the vector of data for the original problem and d_j are randomly generated perturbation vectors. The corresponding solution subset S is given by:

$$S = \{h(p_j), j = 1, \ldots, m\}.$$

Solution subsets may also be generated by simultaneously varying heuristics and perturbing problems.

When local search is based on problem perturbation neighborhoods, it is necessary to evaluate the objective function using original problem data in order for the solution to have meaning. That is, neighboring solutions are generated by applying the base heuristic to the perturbed problem, then the solution is evaluated using the original problem data. When this method is applied to scheduling, a sequencing of jobs on machines results.

To determine the actual schedule (i.e., a schedule which is feasible to the original problem), it is usually a simple matter to build the actual schedule from the job sequence using original processing times and precedence constraints.

3.1. *Problem Space Based Search Neighborhoods*

In this section, several properties of problem space based local search are discussed. Since local search requires specification of a neighborhood of solutions, a desirable property of problem space solution encodings is that a natural neighborhood metric can be defined. Let p be the vector of problem data and d be a random vector of perturbations. A problem space neighborhood is defined by the ball:

$$N_p = \{\text{all } p + d \text{ such that } \| d \| < \epsilon\},$$

There exists a corresponding neighborhood of solutions:

$$N_s = \{\text{all } h(q) \text{ such that } q \text{ is in } N_p\},$$

Several additional considerations regarding problem space based local search can be identified.

First, every solution produced by the method is a solution generated by the base heuristic. Presumably, this base heuristic has been chosen to be well suited to the problem (i.e., it is a "good" heuristic). Thus if only slight perturbations of the original problem are used, it seems reasonable that "good" solution sequences will be generated by the base heuristic. That is, we expect solutions in the neighborhood of the original problem to be good solutions. Thus the search can be restricted to an area where promising solutions should be concentrated.

A second consideration is that the base heuristic may yield the same solution sequence before and after the problem is perturbed. This is especially likely when small perturbations are used. The disadvantage of this phenomenon is that progress of the local search algorithm may be slowed by repeatedly generating the same solution. The algorithm thus requires "tuning" so that the perturbations are large enough to generate new solutions, but not so large as to generate poor solutions. The possibility of turning this phenomenon to an advantage exists. It permits "pseudo simulated annealing" in that a newly generated problem can be accepted as the incumbent even if it produces the same solution. In this case the algorithm will make progress through the problem based search space even though it produces the same solution.

Another advantage of problem space is that only a single heuristic is required. If the heuristic is to be "perturbed" rather than the problem, families of heuristics must be defined, and a "heuristic perturbation strategy" identified. In some problems, defining families of heuristics to form neighborhoods may prove difficult thus limiting the generic applicability of the method. Conversely, in §4 it will be shown that JSP is amenable to neighborhoods defined by families of heuristics.

Also worthy of note is that the problem space based neighborhood is easily embedded in genetic algorithms (as well as other probabilistic search procedures). A major obstacle in the application of genetic algorithms to combinatorial problems has been the difficulty of producing a well-defined crossover operator. Application of GA's to combinatorial problems usually relies on a solution sequence encoding. The application of standard crossover to such encodings invariably yields infeasible offspring solutions. For example in the JSP, if half of a sequence is taken from each parent, it is highly unlikely that the offspring solution will be a valid schedule. To overcome this problem, unwieldy crossover operators are usually defined which guarantee legal solutions, but also lose many of the inheritance properties of standard crossover (cf. Greffenstette 1987). By operating in problem space, standard crossover operators are trivially constructed and applied: form an offspring problem by taking half of the processing times from one parent, and half

from the other. The base heuristic is then applied to the offspring problem to produce a solution guaranteed to be feasible.

The method by which the problem data are to be perturbed merits consideration. For our implementation on the job shop scheduling problem in §4.3 we simply add uniformly distributed random numbers to the data. In other problems, other means of perturbing the problem data may prove useful. A good example is when "rescaling" the problem data causes the heuristic to produce a different solution. An example is the case of the space filling curve heuristic for the planar TSP (Bartholdi and Platzman 1985). If the cities are simply rotated or translated, the heuristic will produce different solutions. Thus the heuristic can be forced to generate alternative solutions without changing the fundamental nature of the problem.

A final important property of problem space is its content. Under quite general conditions it can be shown that all possible solutions (including of course the optimal solution) are contained in problem space. Sufficient conditions for this to be true exist if: (1) the base heuristic can be "reversed" to find a set of problem data that will cause the base heuristic to generate a given solution, and (2) the method used to perturb the problem can (perhaps after repeated application) generate any instance (of a given size) of problem data. Under these conditions, it is a simple matter to prove that all solutions can be reached by a local search of problem space. By (1), for every solution s there is a set of problem data p_s which, when plugged into the base heuristic, will yield s. By (2), the perturbation method is capable of generating p_s.

3.2. *Heuristic Space Based Search Neighborhoods*

Search spaces can also be generated by defining a space of heuristics. The key to successful implementation of this method is the ability to develop parametrized versions of problem specific heuristics, $h(\pi; p)$ where π parametrizes the heuristic. Local search can then be applied to this parameter space. The parametrization of the problem specific heuristic is, by nature, a problem specific endeavor. Thus heuristic space search is not as readily extended to other problems as are the problem space based methods. Fortunately, a number of parametrizations of dispatching rule heuristics for the JSP can be easily defined. One example involves defining a new dispatching rule as the weighted linear combination of dispatching rules. Define the rule used to prioritize jobs at each scheduling decision by:

$$\text{RULE}_i = -\pi_1(P_i) + \pi_2(W_i) + \pi_3(O_i)$$

where i indexes operations, P is processing time, W is work remaining, and O is number of operations remaining. This example rule is a hybrid of SPT, MWR, and MOR when π_j are all greater than zero and the job with maximum RULE_i is selected. The parameter space (π_1, π_2, π_3) serves as a basis for local search. A second example is to divide the scheduling decisions into groups or "windows" according to their sequence in the algorithm. One of several possible dispatching rules is assigned to each window. For example, SPT may be used to dispatch the first 10 operations, MWR the next 10, etc. With W windows, the heuristic is parametrized by $(\pi_1, \pi_2, \ldots, \pi_W)$ where each π_j parametrizes a set of predefined dispatching rules, e.g., $\pi_j \in$ [SPT, LPT, MWR, MOR]. A similar parametrization is used successfully in later sections.

Defining neighborhoods in parameter space will depend on the nature of the parametrization, but is usually a straightforward endeavor. When the parameter space is defined in R^N as in the first example above, neighborhoods can be constructed using standard distance metrics. In the second example, an obvious neighborhood can be generated by changing only one rule (π_j) in the string. That is, the neighborhood consists of all rule strings differing from the incumbent by only a single element in the string.

It is also possible to combine heuristic and problem space methods. As an example,

in §5 heuristic space is first searched to find a "best heuristic." Then, based on this heuristic, a search in problem space is conducted to further refine the solution.

4. Search Heuristics for Job Shop Scheduling

In this section, search heuristics are developed for the classic job shop scheduling problem (JSP). Perhaps the most desirable properties of these heuristics are their basic simplicity and the ease with which they can be developed and coded. Another advantage is that they can be easily applied to any scheduling objective (e.g., makespan, mean flow time, due date based objectives, etc.). The first search algorithm is heuristic space based, while the second searches in problem space.

4.1. *The Base Heuristic*

The fast base heuristic that will be used to generate the search space is based on a hybrid version of the well-known concepts of active and nondelay schedule generation and priority dispatching rules (Giffler and Thompson 1960). Giffler and Thompson proposed two methods by which schedules can be generated (and partially enumerated) in a tree structure. A node in the tree corresponds to a partial schedule and each arc from a node represents the scheduling of one member of the set of currently "schedulable" job operations. Schedulable job operations are unscheduled operations with scheduled immediate predecessors. Thus each node represents a scheduling decision, the arcs represent possible choices, and leaves of the tree are the enumerated set of schedules. Two methods were proposed by Giffler and Thompson, "nondelay," and "active" schedule generation. The set of nondelay schedules N have the property that no machine remains idle if a job is available for processing. The set of active schedules A have the property that no operation can be started earlier without delaying another job. Active schedules form a much larger set and include nondelay schedules as a subset. For makespan problems it can be easily shown that the optimal schedule is in the set A, while no such guarantee exists for N. On the other hand, previous research has found that "nondelay dispatching is a better basis for heuristic schedule generation than active dispatching" (Baker 1974).

In the present research, a hybrid method has been developed in which the set of possible schedules is a subset of A which includes N. The algorithm is parametrized by $\delta \in [0, 1]$ such that when $\delta = 0$ the set of schedules considered is N, while $\delta = 1$ results in A. As δ increases from 0 the set of schedules increases in cardinality. In an active schedule, a machine is allowed to remain idle on the chance that a "more critical" job will soon become available. The parameter δ can be thought of as defining a bound on the length of time a machine is allowed to remain idle. Details of the hybrid algorithm, a proof that $\delta = 0$ yields N and $\delta = 1$ yields A, and a proof that the algorithm is $O(n^2)$ on the number of jobs are relegated to Appendix A.

4.2. *Heuristic Based Search Space for the JSP*

Given a dispatching rule, the hybrid scheduler will generate a schedule for an instance of JSP. A space of solutions can be defined by a set of dispatching rules. This concept can be extended by breaking the time line into several (W) time windows, and then assigning a dispatching rule to each window. Alternatively, one can assign one dispatching rule to the first X scheduling decisions, a second rule to the next X decisions, etc. The set of possible ways to assign dispatching rules to windows defines the heuristic space. The corresponding solution space is the set generated by applying these heuristics to the problem. Parameters which determine the size of the heuristic space are the number of dispatching rules included (D) and the number of windows (W). That is, with D different dispatching rules, and W windows, there are D^W different elements of the heuristic space. It is more than likely that two different elements of heuristic space will yield the same

solution. The larger the number of windows, the more repeat solutions are likely to exist. Further, a larger search space is obviously harder to search, thus a rule of diminishing returns applies in selecting an appropriate number of windows. As discussed in §5, experiments with 5, 10, and 20 windows and six dispatching rules (SPT, LPT, MWR, LWR, MOR, LOR) have been conducted.

The value of δ chosen in the schedule generation algorithm will also affect the success of the search. For δ close to zero, the search will be restricted to schedules with little or no unnecessary idle time. These schedules will tend to be better on average than those produced by a higher value of δ. However, by restricting the possible set of schedules, optimal or some near optimal solutions may be excluded. In the present research, values of δ in the range [0, 0.1] are investigated.

Several search methods could be employed to search the solution space. Possibilities include hill climbing, steepest descent, simulated annealing, genetic algorithms, etc. In the present paper simple hill climbing is employed as follows:

Initialize: create an "incumbent heuristic" by assigning a randomly selected dispatching rule to each window.

Iterate through the following steps L times:

1. Select a window at random.
2. Assign a randomly selected rule (different from the current incumbent rule) to this window.
3. Apply the new heuristic, obtain the solution and its makespan.
4. IF the new makespan is equal to, or smaller than that of the incumbent solution, update the incumbent heuristic to include the new rule.
   ELSE maintain the previous incumbent heuristic.
5. Go to 1.

4.3. *A Problem Based Search Space for the JSP*

A second search space can be defined for scheduling problems by perturbing the problem data. In particular, letting i, j index operation j of job i, the processing times of operations P_{ij} are perturbed by a random amount to yield "dummy" processing times DP_{ij}. These dummy processing times are used at each decision point in the application of the dispatching rule. The hybrid generator with the SPT dispatching rule is used as the fast base heuristic such that the schedulable operation with minimum DP_{ij} is scheduled next. When operations are actually scheduled, the real processing times P_{ij} are used to construct the schedule. The dummy processing times are created by adding a randomly generated, uniformly distributed amount with mean zero to the real processing times:

$$DP_{ij} = P_{ij} + \bar{U}_{ij} \quad \text{where the } U_{ij} \text{ are iid Uniform } (-\theta, \theta).$$

The parameter θ controls the size of the perturbations. The values of δ and θ determine the size and content of the search neighborhood. When $\delta = 1$ and θ approaches infinity, the search neighborhood contains all active schedules. Conversely, $\delta = 0$ and small θ yields a search neighborhood consisting of nondelay, SPT schedules with ties broken arbitrarily.

A variety of local search techniques including simulated annealing or genetic algorithms could be employed. In an attempt to keep our focus on the search spaces rather than the search technique, a simple variant of hill climbing is used. Intuitively, when the dummy processing times diverge from the real processing times, the quality of solutions investigated will degrade. A variant of hill climbing which might be termed "pseudo-steepest descent" helps to overcome this problem. The method investigates several neighbors of the incumbent solution before selecting a new incumbent, and proceeds as follows:

Initialize: Set Best$M = \infty$. Set $IDP_{ij} = P_{ij}$ for all i, j.

Iterate S times through steps (A)–(C):

(A) Iterate N times through steps 1–4:

1. Set $DP_{ij} = IDP_{ij} + U_{ij}$ for all i, j where U_{ij} are iid $U(-\theta, \theta)$.
2. Apply heuristic with SPT based on DP_{ij}.
   Obtain solution and makespan M.
3. IF $M <$ Best$M$.
   THEN set Best$M = M$, set $BDP_{ij} = DP_{ij}$ for all i, j.
4. Go to 1.

(B) Set $IDP_{ij} = BDP_{ij}$ for all i, j.

(C) Go to (A).

The algorithm searches the neighborhood of the incumbent solution defined by IDP_{ij} by iterating through steps 1 to 4 N times. After N neighbors have been investigated, the best solution is determined, and the IDP_{ij} are reset to reflect this best solution. This procedure is repeated S times resulting in the generation of a total of $L = SN$ schedules. The search method just described requires specification of N, and the length of the search, L. Here, L is defined as the number of applications of the base heuristic (i.e., the number of schedules generated). Experiments described in §5 have been conducted to suggest appropriate values for these parameters.

The proposed method has much in common with random and probabilistic dispatching rules that have been previously proposed (a summary of these methods may be found in Baker 1974). When θ is close to zero, the effect of the perturbations is to randomly break ties for schedulable jobs with the same processing times (jobs tied for the shortest processing time). As θ approaches infinity, the effect is to randomly select one of the schedulable jobs. This latter method is equivalent to the random dispatching rule (Baker 1974). When θ is between 0 and infinity, jobs with shorter processing times will be favored. This is similar in principle to probabilistic dispatching rules in which the job to schedule is selected probabilistically with higher probability given to jobs ranked more highly by the dispatching rule. For example, Baker (1974) discusses a probabilistic dispatching rule in which the list of schedulable jobs is ranked by MWR and probabilities assigned geometrically. The probabilistic heuristic is applied repeatedly to generate a population of solutions.

It is important to understand the distinction between probabilistic dispatching and the proposed method. The fundamental distinction is that the proposed method can be used to generate solution spaces and neighborhoods to which local search can be applied. Conversely, probabilistic dispatching provides only a means to generate alternative solutions. Thus the distinction is that the proposed method entails local search as opposed to picking the best among probabilistically generated solutions. Differences of a less fundamental nature can also be identified. Probabilistic dispatching requires that the list of schedulable jobs be rank ordered at each decision point ($O(n \log n)$), whereas the proposed method requires only that the first job on the list (the job with shortest processing time) be identified ($O(n)$). The proposed method can thus be applied with less computational effort. A final distinction is that the probability of a job being selected will be based on its actual processing time in the proposed method rather than on its rank in the SPT sorted list of schedulable jobs.

The problem space search method can be easily generalized to other scheduling problems and other base heuristics. For example, in a scheduling problem with due date based criteria, a base heuristic using slack time as the dispatching rule might be used. Perturbed processing times could be used in computing slack time when the dispatching rule is applied. As another example, MOR could be used as a base dispatching rule. Each operation (i, j) would be assigned a "dummy number of operations" DNO_{ij} perhaps generated as $DNO_{ij} = 1 + U(-\theta, \theta)$.

4.4. *Extension to Other Local Search Methods*

Although simple local search methods based on hill climbing are used in this paper, it should be noted that the proposed JSP search spaces are ideally suited for simulated annealing and genetic algorithms. Simulated annealing could be directly applied using either (problem or heuristic) search space. Genetic algorithms are also easily constructed. In problem space an encoding simply entails a listing of the problem data, in this case processing times (precedence relations are assumed fixed). By combining processing times from two parent problems via standard crossover and applying the heuristic, a feasible schedule is guaranteed. Initial populations and a mutation operator may also be constructed in a straightforward manner.

A number of chromosomal representations of solutions can be developed for genetic algorithms in heuristic space. The obvious method is to break the time line into time windows and assign a dispatching rule to each time window as proposed above. The list of dispatching rules assigned to each window serves as an encoding for a GA. Since a solution is represented by a time ordered list of dispatching rules, standard GA crossover operators may be applied. A second alternative is to assign a dispatching rule to each operation such that the rule is applied to the scheduling decision following completion of that operation. A third possible encoding is based on the idea of weighted dispatching rules. The set of weights assigned to each dispatching rule form a vector, and thus a chromosomal representation of a solution. GA's can be applied to this representation to search for a problem specific universal dispatching rule. All of these proposals for GA's overcome previously encountered problems in the application of GA's to "sequencing" problems, namely problems of feasibility and the need for unwieldy crossover operators.

5. Test Results for Job Shop Scheduling with Makespan Objective

In this section the results of initial testing of the algorithms described in §4 are presented. All testing was performed on JSP problems with minimum makespan as the objective. While makespan has been criticized as an unrealistic objective, it was chosen for comparison purposes because of the availability of good alternative heuristics. Also a few test problems with known optimal solution have been reported in the literature. It is reiterated that a principle advantage of the proposed method is its applicability to scheduling problems with any objective, and that the choice of makespan serves primarily to provide a challenging test.

Two sets of experiments were run. First, experiments were conducted to determine appropriate values of algorithm parameters. Next, the performance of both methods is tested and compared to a previously proposed method, the shifting bottleneck heuristic of Adams et al. (1988).

5.1. *Algorithm Parameter Tuning*

Since both the "heuristic space" and "problem space" algorithms involve several parameters, the first set of experiments was aimed at determining appropriate values for these parameters. For the heuristic space method, parameters investigated are δ (the hybrid schedule generator parameter), W (the number of windows), and L (the length of the search measured as the number of applications of the hybrid schedule generator). For the problem space method, four parameters were investigated, δ, L (both as above), θ (the size of the perturbations), and N (the number of neighbors investigated before a new incumbent is chosen).

Five test problems were generated each consisting of 10 machines and 50 jobs. Each job consisted of 10 operations, one operation to be performed on each machine. Processing times were generated uniformly between 1 and 100. In a first attempt, the machine precedence constraints for each job were generated randomly. This method resulted in

rather unchallenging problems which are referred to as "easy" problems in the sequel. As an alternative, a simplified version of a method proposed by Muth and Thompson (1963) to generate challenging problems was adopted. Each job must pass first through machines 1 through 5, then through machines 6 to 10. The ordering on each of the two sets of five machines was generated randomly. Problems generated by this method will be referred to as "hard" problems. Portable Fortran code to reproduce all test problems will be provided by the authors for a limited time upon request.

For each method, full factorial experimental designs aimed at parameter tuning were run as shown in Table 1. Two replicates were run in each cell by providing different random number seeds to the algorithms.

The heuristic space experiment was analyzed by ANOVA with the five problems as blocking factors. The number of windows was highly significant ($p = 0.0000$), δ was marginally significant ($p = 0.054$), and L was insignificant ($p = 0.58$). No interaction effects were significant. The results indicated that 20 windows and $\delta = 0.05$ or 0.1 were the best combination tried. Although L was statistically insignificant, treatment mean makespan was lower for $L = 2{,}000$. The important conclusion from this experiment is that 20 windows yields best results. Since L controls the length of search, the obvious conclusion is that L should be as big as possible. However, the results indicated that most of the improvement in the solution is attained within 2,000 iterations.

Results from the problem space experiment were analyzed in a similar fashion. Both θ and L were highly significant ($p = 0.0000$) while δ was again marginally significant ($p = 0.056$). Small perturbations ($\theta = 10$ or 20), $\delta = 0.05$ or 0.1 and $L = 2{,}000$ Schedules proved to be the best combinations.

5.2. *The Shifting Bottleneck Heuristic*

The solutions generated by the heuristics developed in §4 will be compared with solutions from the shifting bottleneck (SB) heuristic first described in Adams et al. (1988). The SB implementation we used was graciously provided to us by William Cook. Details of this implementation may be found in Applegate and Cook (1991). The SB heuristic was chosen as a comparative base in this paper for the simple reason that it has been documented to produce high quality solutions. Thus, by comparison with SB, the quality of solutions produced by our heuristics can be judged. However, by no means can the results be used to infer "which method is best." The SB and proposed heuristics are fundamentally different in nature. That is, we would be "comparing apples and oranges." The local search methods we describe can be run for as many or as few iterations as desired. Clearly the more iterations, the better the solution will be.

The Applegate and Cook implementation of SB worked well in general; however it was unable to solve two of the 10×50 problems. The precise reason for this is unknown to us. However we can report that in an earlier version of the paper, we used our own (less sophisticated) version of the SB heuristic. We encountered problems both with cycling and with the branch and bound algorithm used to solve one-machine subproblems within the SB heuristic. On larger (10×50) problems our implementation usually failed.

TABLE 1

Algorithm Parameter Levels Included in the Full Factorial Tuning Experiment

Problem Space Experiment				Heuristic Space Experiment			
δ	0	0.05	0.1	δ	0	0.05	0.1
θ	10	20	50	θ	10	20	50
N	100	200		L	1000	2000	
L	1000	2000					

5.3. *Test Results*

Various versions of the proposed heuristics and the SB were run of a suite of test problems. The test problems include two well-known small problems from Muth and Thompson (1963), one a 10 × 10 (ten machines, ten jobs), and the second a 5 × 20. These problems were also tested in Adams et al. The remaining problems were generated by the methods discussed above. Five problems were generated in each of the following classes: 10 × 20 "hard", 15 × 20 "hard", 10 × 50 "hard", and 10 × 50 "easy". Testing was performed on the heuristic space, problem space, and a combined method. The combined method uses the best heuristic from the search of heuristic space as the base heuristic for a search in problem space. Values of $\delta = 0$ and 0.1 were tested for each method. The problem space heuristics were run for a total of $L = 2{,}000$ iterations ($N = 200$). The heuristic space methods were run for $L = 5{,}000$ iterations and the best solution after $L = 50$ and $L = 5{,}000$ iterations reported. All algorithms were written in Fortran and run on a Sun system 4/280. Experimental results appear in Tables 2 and 3.

TABLE 2

Comparison of Heuristic Results (Makespan) on Two Problems from Muth and Thompson (MT) and on Randomly Generated Test Problems

Problem	SB	CS0	CS10	PS0	PS10	HSL0	HSL10	HSS0	HSS10
10 × 10 MT	952	1056	1006	984	976	1009	1006	1052	1037
5 × 20 MT	1228	1191	1194	1193	1186	1244	1206	1267	1222
10 × 20 "HARD"	1640	1547	1575	1551	1518	1557	1517	1641	1555
	1643	1651	1601	1636	1599	1615	1625	1664	1716
	1706	1626	1626	1604	1617	1619	1636	1672	1686
	1708	1686	1642	1653	1646	1748	1676	1801	1783
	1561	1632	1636	1658	1637	1605	1620	1662	1741
15 × 20 "HARD"	1913	1949	1953	1936	1917	2020	1937	2104	1971
	1983	1851	1808	1821	1792	1839	1850	1880	1856
	2099	2038	2026	2024	2027	2081	2016	2141	2144
	1964	1918	1920	1928	1953	1979	1933	2024	2048
	1935	1972	1940	1985	2010	1981	1994	2003	2088
10 × 50 "HARD"	3423	3293	3303	3309	3346	3322	3375	3458	3419
	3508	3386	3376	3278	3389	3370	3405	3522	3501
	unkn	3472	3487	3494	3485	3453	3516	3595	3600
	3280	3189	3200	3229	3260	3232	3194	3424	3367
	unkn	3174	3225	3181	3257	3233	3225	3258	3262
10× 50 "EASY"	2924	2924	2924	2924	2929	2924	2924	2924	2924
	2794	2794	2794	2794	2809	2794	2794	2794	2805
	2852	2852	2852	2852	2855	2852	2852	2852	2859
	2843	2843	2843	2843	2899	2843	2843	2843	2853
	2823	2823	2823	2823	2823	2823	2823	2824	2859

SB: Shifting Bottleneck.
CS0: Combined space, $\delta = 0$.
CS10: Combined space, $\delta = 10$.
PS0: Problem space, $\delta = 0$.
PS10: Problem space, $\delta = 10$.
HSL0: Heuristic space, 5000 iter, $\delta = 0$.
HSL10: Heuristic space, 5000 iter, $\delta = 10$.
HSS0: Heuristic space, 50 iter, $\delta = 0$.
HSS10: Heuristic space, 50 iter, $\delta = 10$.

TABLE 3

Comparison of Heuristic Results (Makespan) on Test Problems: Percent from Best Solution

Problem	SB	CS0	CS10	PS0	PS10	HSL0	HSL10	HSS0	HSL10
10 × 10 MT	0.0	3.4	2.5	6.0	5.7	10.5	8.9	10.9	5.7
5 × 20 MT	3.5	0.6	0.0	4.9	1.7	6.8	3.0	0.4	0.7
10 × 20 "HARD"	8.1	2.0	3.8	2.2	0.1	2.6	0.0	8.2	2.5
	2.8	3.3	0.1	2.3	0.0	1.0	1.6	4.1	7.3
	6.4	1.4	1.4	0.0	0.8	0.9	2.0	4.2	5.1
	4.0	2.7	0.0	0.7	0.2	6.5	2.1	9.7	8.6
	0.0	4.5	4.8	6.2	4.9	2.8	3.8	6.5	11.5
15 × 20 "HARD"	0.0	1.9	2.1	1.2	0.2	5.6	1.3	10.0	3.0
	10.7	3.3	0.9	1.6	0.0	2.6	3.2	4.9	3.6
	4.1	1.1	0.5	0.4	0.5	3.2	0.0	6.2	6.3
	2.4	0.0	0.1	0.5	1.8	3.2	0.8	5.5	6.8
	0.0	1.9	0.3	2.6	3.9	2.4	3.0	3.5	7.9
10 × 50 "HARD"	3.9	0.0	0.3	0.5	1.6	0.9	2.5	5.0	3.8
	7.0	3.3	3.0	0.0	3.4	2.8	3.9	7.4	6.8
	unkn	0.6	1.0	1.2	0.9	0.0	1.8	4.1	4.3
	2.9	0.0	0.3	1.3	2.2	1.3	0.2	7.4	5.6
	unkn	0.0	1.6	0.2	2.6	1.9	1.6	2.6	2.8
10 × 50 "EASY"	0.0	0.0	0.0	0.0	0.2	0.0	0.0	0.0	0.0
	0.0	0.0	0.0	0.0	0.5	0.0	0.0	0.0	0.4
	0.0	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.2
	0.0	0.0	0.0	0.0	2.0	0.0	0.0	0.0	0.4
	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.3
Ave %	2.8	1.4	1.0	1.4	1.5	2.5	1.8	4.6	4.3

SB: Shifting Bottleneck.
CS0: Combined space, $\delta = 0$.
CS10: Combined space, $\delta = 10$.
PS0: Problem space, $\delta = 0$.
PS10: Problem space, $\delta = 10$.
HSL0: Heuristic space, 5000 iter, $\delta = 0$.
HSL10: Heuristic space, 500 iter, $\delta = 10$.
HSS0: Heuristic space, 50 iter, $\delta = 0$.
HSS10: Heuristic space, 50 iter, $\delta = 10$.

The results indicate that all of the proposed methods can produce solutions which are competitive with the shifting bottleneck procedure. Of particular note is the performance of the problem space methods. The combined space method appears to yield little improvement over pure problem space especially when the added computation is considered. The heuristic space method with 5,000 iterations is also beaten by the problem space method (with only 2,000 iterations).

6. Conclusions

This paper presents a simple and generic methodology for developing search spaces and simple local search algorithms for a wide variety of sequencing problems. The key to the method is the incorporation of problem-specific information into the definition of the search space through the use of known heuristics and the heuristic, problem (h, p) solution encoding. The result is to cluster good solutions close together, thus making it easier to perform local search. Probabilistic search methods such as simulated annealing and genetic algorithms can be applied directly to the search spaces.

The usefulness of the method was demonstrated by developing heuristics for job shop scheduling. In this realm, the method can be easily applied to scheduling problems with any objective. The quality of solutions produced by the proposed heuristics was demonstrated on test problems with makespan objective. These results are very encouraging when compared to the shifting bottleneck procedure.

Appendix: Hybrid Scheduling Algorithm A1

The following scheduling algorithm is a modification of the nondelay and active schedule generation algorithms of Giffler and Thompson (1960). The parameter $\delta \in [0, 1]$ parametrizes the algorithm. When $\delta = 0$, A1 produces nondelay schedules. When $\delta = 1$, A1 produces active schedules. For values of δ between 0 and 1, A1 produces schedules from a subset of active schedules which includes all nondelay schedules. The algorithm is equivalent to active schedule generation except for a modification in step 2. The only other modification is that the algorithm uses a separate list of "schedulable operations" for each machine, whereas Giffler and Thompson's algorithms are based on a single list. This modification is included only because it improves programming efficiency. The notation used is similar to that of Baker (1974).

Notation

m indexes machines.
(i, j) indexes the jth operation of job i.
$P(i, j)$ is the processing time required for (i, j).
$M(i, j)$ is the machine required for (i, j).
$N(i)$ is the number of operations to complete job i.
$I(m)$ is the list of operations currently schedulable on m.
$= \{\text{unscheduled } (i, j)\text{: } M(i, j) = m \text{ and } (i, j-1) \text{ scheduled}\}$.
I is the set of all schedulable $(i, j) = \{I(1), I(2), \ldots, I(m)\}$.
$C(m)$ is the completion time of the last operation scheduled (thus far) on machine m.
$T(i, j)$ is the scheduled start time of (i, j).
$\sigma(i, j)$ = earliest start time of (i, j),
$= \text{MAX}\{C[M(i, j)], T(i, j-1) + P(i, j-1)\}$.
$\phi(i, j)$ = earliest completion time of $(i, j) = \sigma(i, j) + P(i, j)$.
$\sigma^*(m) = \text{MIN}_{i \in I(m)}\, \sigma(i, j)$ = earliest start time on machine m.
$\sigma^* = \text{MIN}_m\, \sigma^*(m)$ = earliest start time of all schedulable jobs.
$\phi^*(m) = \text{MIN}_{i \in I(m)}\, \phi(i, j)$ = earliest completion time on machine m.
$\phi^* = \text{MIN}_m\, \phi^*(m)$ = earliest completion time of all schedulable jobs.
$\delta \in [0, 1]$.

ALGORITHM A1. *Generalized Schedule Generator (assumes all jobs available at time* 0).

0. Initialize:

$$I(m) = \{(i, 1)\text{: } M(i, 1) = m\} \qquad \text{for all jobs } i,$$

$$\sigma(i, 1) = 0 \qquad \text{for all } i,$$

$$\phi(i, 1) = P(i, 1) \qquad \text{for all } i,$$

$$C(m) = 0 \qquad \text{for all } m,$$

$$\sigma^*(m) = 0 \qquad \text{for all } m,$$

$$\sigma^* = 0,$$

$$\phi^*(m) = \underset{i \in I(m)}{\text{MIN}}\, \phi(i, 1),$$

$$\phi^* = \underset{m}{\text{MIN}}\, \phi^*(m).$$

1. Schedule next on machine m with earliest completion time (Break ties arbitrarily):
   Find m such that $\phi^*(m) = \phi^*$.
2. Find the set S of possible operations to schedule:

$$S = \{(i, j) \in I(m)\text{: } \sigma(i, j) \le \sigma^*(m) + \delta(\phi^*(m) - \sigma^*(m))\}.$$

   (This is the main modification to the active scheduling algorithm.)
3. Choose an operation (s, j) from S to schedule next. Typically this choice is made by application of a dispatching rule.

$$T(s,j) = \sigma(i,j),$$

$$C(m) = T(s,j) + P(s,j).$$

Remove (s,j) from $I(m)$,

4. Update σ and ϕ for operations in $I(m)$:

$$\sigma(i,j) = \text{MAX}\,(\sigma(i,j), C(m)) \qquad \text{for all } (i,j) \in I(m),$$

$$\phi(i,j) = \sigma(i,j) + P(i,j) \qquad \text{for all } (i,j) \in I(m).$$

Update $\sigma^*(m)$, $\phi^*(m)$, σ^*, and ϕ^* if necessary.

5. If job s not complete, add its next operation to the list of schedulable jobs for its next machine:

$$\text{IF} \quad j < N(s) \quad \text{THEN} \quad \text{add } (s, j+1) \text{ to } I(M(s,j+1)),$$

$$\sigma(s,j+1) = \text{MAX}\,(T(s,j) + P(s,j), C(M(s,j+1))),$$

$$\phi(s,j+1) = \sigma(s,j+1) + P(s,j+1).$$

Update $\sigma^*(M(s,j+1))$, $\phi^*(M(s,j+1))$, σ^*, and ϕ^* if necessary.

6. If operations remain to be scheduled, go to 1.

THEOREM. *Algorithm* A1 *produces nondelay schedules when* $\delta = 0$; *and active schedules when* $\delta = 1$.

PROOF. When $\delta = 1$, Algorithm A1 reduces to the active schedule generation algorithm. It is somewhat more difficult to show that $\delta = 0$ produces nondelay schedules. By definition of a nondelay schedule, it must be shown that in a schedule produced by A1 ($\delta = 0$), no operation is available for processing on a machine m during an idle period on machine m.

Consider an arbitrary idle period on a machine (say m) in a schedule produced by A1 ($\delta = 0$). Let (s,j) be the operation which succeeds the idle period. To show that no operation is available during the idle period it is necessary to consider only operations requiring machine m.

Consider the point during A1 at which (s,j) is scheduled, and divide all operations requiring m into 3 sets:

$A = \{\text{operations previously scheduled on } m\}$,

$B = \{\text{currently schedulable operations requiring } m\} = I(m)$,

$C = \{\text{operations requiring } m \text{ with a predecessor on the list of schedulable operations } I\}$.

Case A. Since operations in A have been scheduled and completed on m prior to the idle period, they are clearly not available during the idle period.

Case B. By steps 2 and 3 of A1:

$$T(s,j) = \sigma(i,j) = \sigma^*(m) \le \sigma(i,j) \qquad \text{for all} \qquad (i,j) \in I(m).$$

Thus no operation in B is available before $T(s,j)$, and clearly cannot be available during the idle period which ends at $T(s,j)$.

Case C. Let $(i, j+k)$ be an operation in C and (i,j) its predecessor which is currently schedulable (in I). We have:

$\sigma(i,j+k) \ge \phi(i,j)$ (must complete (i,j) before starting $(i,j+k)$),
$\phi(i,j) \ge \phi^*$ (by definition of ϕ^*),
$\phi^* = \phi^*(m)$ (by step 1 of A1),
$\phi^*(m) \ge \sigma^*(m)$ (earliest completion time must be $\ge$ earliest start time),
$\sigma^*(m) = T(s,j)$ (by steps 2 and 3 of A1).
Thus $\sigma(i,j+k) \ge T(s,j)$ for all $(i,j+k) \in C$.
So no job in C is available during the idle period.

Assuming each job must be processed on each machine exactly once, Algorithm A1 runs in $O(N^2)$ where N is the number of jobs to be scheduled. This can be shown as follows. Letting M be the number of machines, there are NM operations and NM scheduling decisions to be made (NM iterations through steps 1 to 6). Step 2 requires one pass through all jobs in $I(m)$. Step 4 also requires one pass through $I(m)$. On average there will be N/M operations in $I(m)$. Thus the complexity of A1 is:

$$(NM)(2)(N/M) = 2N^2.$$

References

ADAMS, J., E. BALAS AND D. ZAWACK, "The Shifting Bottleneck Procedure for Job Shop Scheduling," *Management Sci.*, 34, 3 (1988).

APPLEGATE, D. AND W. COOK, "A Computational Study of the Job-Shop Scheduling Problem," *ORSA J. Comput.*, 3, 2 (1991), 149–156.

BAKER, K. R., *Introduction to Sequencing and Scheduling*, John Wiley and Sons, New York, 1974.

BARTHOLDI, J. J. AND L. K. PLATZMAN, "Heuristics Based on Space Filling Curves for Combinatorial Problems in Euclidian Space," *Management Sci.*, 34, 3 (1985), 291–305.

BONOMI, E. AND J. LUTTON, "The *N*-city Traveling Salesman Problem: Statistical Mechanics and the Metropolis Algorithm," *Siam Rev.*, 26, 4 (1984), 551–568.

CERNY, V., "Thermodynamical Approach to the Traveling Salesman Problem: Statistical Mechanics and the Metropolis Algorithm," *J. Optim. Theory Appl.*, 41 (1985), 41–51.

COMMITTEE ON THE NEXT DECADE OF OPERATIONS RESEARCH (CONDOR), "Operations Research: The Next Decade," *Oper. Res.*, 36, 4 (1988).

DAVIS, L., "Job Shop Scheduling with Genetic Algorithms," In J. Grefenstette (Ed.), *Proc. Internat. Conf. Genetic Algorithms and Their Applications*, Carnegie Mellon University, 1985.

ECK, B. AND M. PINEDO, "Good Solutions to Job Scheduling Problems Via Tabu Search," Presented at Joint ORSA/TIMS Meeting, Vancouver, May 10, 1989.

GIFFLER, B. AND G. L. THOMPSON, "Algorithms for Solving Production Scheduling Problems," *Oper. Res.*, 8, 4 (1960).

GITTINS, J. C., "Bandit Processes and Dynamic Allocation Indices," *J. Ray. Statist. Soc. Ser. B*, 41, 2 (1979), 148–177.

GLOVER, F., "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers and Oper. Res.*, 13, 5 (1986), 533–549.

———, "Tabu Search. Part I," *ORSA J. Comput.*, 1, 3 (1989), 190–205.

———, "Tabu Search. Part II," *ORSA J. Comput.*, 2, 1 (1990), 4–32.

GOLDBERG, D. E. AND R. LINGLE, "Alleles, Loci, and the Traveling Salesman Problem," In J. Grefenstette (Ed.), *Proc. Internat. Conf. Genetic Algorithms and Their Applications*, Carnegie Mellon University, 1985.

GOLDEN, B. L. AND C. C. SKISCIM, "Using Simulated Annealing to Solve Routing and Location Problems," *Naval Res. Logist. Quart.*, 33 (1986), 261–279.

GREFENSTETTE, J. J., "Incorporating Problem Specific Knowledge into Genetic Algorithms," In L. Davis (Ed.), *Genetic Algorithms and Simulated Annealing*, Pitman, London, 1987.

———, R. GOPAL, B. ROSMAITA AND D. VAN GUCHT, "Genetic Algorithms for the Traveling Salesman Problem," In J. Grefenstette (Ed.), *Proc. Internat. Conf. Genetic Algorithms and Their Applications*, Carnegie Mellon University, 1985.

KIRKPATRICK, S., "Optimization by Simulated Annealing: Quantitative Studies," *J. Statist. Physics*, 34, 5/6 (1984), 975–986.

———, C. D. GELATT AND M. P. VECCHI, "Optimization by Simulated Annealing," *Science*, 220 (1983), 671–680.

LAGUNA, M., J. W. BARNES AND F. GLOVER, "Scheduling Jobs with Linear Delay Penalties and Sequence Dependent Setup Costs Using Tabu Search," Presented at Joint ORSA/TIMS Meeting, Vancouver, May 10, 1989.

LIEPINS, G. E., M. R. HILLIARD, M. PALMER AND M. MORROW, "Greedy Genetics," In J. Grefenstette (Ed.), *Genetic Algorithms and Their Applications: Proc. Second Internat. Conf. Genetic Algorithms*, Massachusetts Institute of Technology, Cambridge, MA, 1987.

LUNDY, M. AND A. MEES, "Convergence of an Annealing Algorithm," *Math. Programming*, 34 (1986), 111–124.

MUTH, J. F. AND G. L. THOMPSON, *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

OLIVER, I. M., D. J. SMITH AND J. R. C. HOLLAND, "A Study of Permutation Crossover Operators on the Traveling Salesman Problem," In J. Grefenstette (Ed.), *Genetic Algorithms and Their Applications: Proc. Second Internat. Conf. Genetic Algorithms*, Massachusetts Institute of Technology, Cambridge, MA, 1987.

PANWALKAR, S. S. AND W. ISKANDER, "A Survey of Scheduling Rules," *Oper. Res.*, 23, 1 (1977), 45–61.

PAPADIMITRIOU, C. H. AND K. STEIGLITZ, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

SUH, J. Y. AND D. VAN GUCHT, "Incorporating Heuristic Information into Genetic Search," In J. Grefenstette (Ed.), *Genetic Algorithms and Their Applications: Proc. Second Internat. Conf. Genetic Algorithms*, Massachusetts Institute of Technology, Cambridge, MA, 1987.

TAILLARD, E., "Parallel Taboo Search Technique for the Jobshop Scheduling Problem," Working Paper ORWP 89/11 Département de Mathematiques, Ecole Polytechnique Fédérale De Lausanne, CH-1015 Lausanne, Switzerland, 1989.

WHITLEY, D., T. STARKWEATHER AND D. FUQUAY, "Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator," *Proc. Third Internat. Conf. Genetic Algorithms*, George Mason University, Morgan Kaufman Publishers, San Mateo, CA, 1989.

WU, S. D., "An Expert System Approach for the Control and Scheduling of Flexible Manufacturing Cells," Unpublished Ph.D. Dissertation, The Pennsylvania State University, 1987.