RANDOM KEYS GENETIC ALGORITHM FOR SCHEDULING: UNABRIDGED VERSION

Bryan Norman
and
James C. Bean
Department of Industrial & Operations Engineering
The University of Michigan
Ann Arbor, Michigan 48109-2117

Technical Report 95-10

July 1995

Random Keys Genetic Algorithm for Scheduling: Unabridged Version

Bryan A. Norman and James C. Bean

Department of Industrial and Operations Engineering

University of Michigan

Ann Arbor, MI 48109

July 20, 1995

Abstract

This paper considers the a scheduling problem to minimize total tardiness given multiple machines, ready times, sequence dependent setups, machine downtime and scarce tools. We develop a genetic algorithm based on random keys representation, elitist reproduction, Bernoulli crossover and immigration type mutation. Convergence of the algorithm is proved. We present computational results on data sets from the auto industry. To demonstrate robustness of the approach, problems from the literature of different structure are solved by essentially the same algorithm.

1 Introduction

In this paper we present a random keys based genetic algorithm to solve scheduling problems. Each job has a prescribed subset of the machines on which it must be processed. The objective is to determine a sequence for placing the jobs on the machines that optimizes a given evaluation measure. A brief description of scheduling terminology is presented below.

The *study horizon* represents the time interval for which a schedule must be developed. The study horizon may be hours, days, weeks, or months. A job's *ready time* is the time that a job is available for processing. If the ready time is zero, then the job is available at the beginning of the study horizon. *Setup time* is the time required to prepare a machine to begin processing a particular job. Setup time may be sequence dependent, meaning that the setup time for a job on a machine depends on the job previously run on the machine. *Machine uptime* represents the time that a machine is available to perform work. A machine may experience periods of downtime due to routine maintenance or breakdowns. Jobs may only be run on a machine during its uptime. A job has *tool constraints* if it requires a special tool or fixture in order to be processed. If different jobs require the same tool, there may be a conflict over tool use. A scheduling problem has *precedence constraints* if some job cannot be started prior to the completion of one or more other jobs. A job's *processing time* is the actual time required to process the job and excludes setup time. The *due date* of a job is the time at which the job must be completed to avoid incurring a penalty.

There are several different measures commonly used for evaluating a schedule. The principle measure considered in this paper is the total tardiness for all jobs. The genetic

algorithm we present below can handle many other objectives such as weighted tardiness and combinations of weighted earliness and tardiness. Throughout this paper, job characteristics are referenced using the following notation:

$$inputs = \begin{cases} p_i & \text{processing time for a job} \\ r_i & \text{ready time for a job} \\ d_i & \text{due time for a job} \end{cases}$$

$$outputs = \begin{cases} c_i & \text{completion time for a job} \\ t_i & \text{tardiness for a job} \end{cases}$$

For a more comprehensive discussion of scheduling problems and terminology see Baker [1974], French [1982], Morton and Pentico [1993], and Pinedo [1995].

In this paper, the principle problem explored consists of actual data from an automaker. There are several factors that complicate the automaker scheduling problems. The jobs have ready times and due dates that range throughout the study horizon. Each job has a particular tooling requirement. Tooling conflicts arise because there is only one copy of each tool and multiple jobs may require that tool. Setup times are introduced into the problem if consecutive jobs on the same machine require different tools. There is processing flexibility because different jobs can be processed on different subsets of the machines although the machines are not identical. The scheduling objective is to minimize the total tardiness of all of the jobs. The number of jobs ranges from 270 to 360. These scheduling problems are clearly NP hard. The scheduling problem to minimize total tardiness for a single machine problem is NP hard even if the following simplifying assumptions are made: all jobs are

ready at time zero, setup times are not sequence dependent, no machine downtime, and no special tool requirements (Du and Leung [1990]).

The vast majority of the papers in the scheduling literature consider only a subset of these problem complexities. Hoitomt et al. [1993], Grabowski et al. [1986], and Widmer [1991] have investigated problems that contain many of the complexities but each of these assumes away two or more of the factors. The only paper that we are aware of that has addressed problems containing all the previously mentioned complexities is the Matchup Scheduling work of Bean et al. [1991]. They propose different priority rules and heuristic methods based on an integer programming formulation of the problem. Section 5 provides a comparison of the Matchup algorithm with the proposed genetic algorithm.

Constructing optimal solution procedures for complicated scheduling problems has proven to be a difficult task. Branch-and-bound algorithms do exist for some difficult problems. For example, Carlier and Pinson [1989], Carlier and Pinson [1990], Carlier and Pinson [1994], Applegate and Cook [1991], and Brucker et al. [1994] have implemented branch-and-bound procedures for the classic job shop problem with the objective of minimizing the makespan. However, these methods require long running times and become impractical as problem size increases. Moreover, for problems with the complexity of the automaker data a branch-and-bound approach quickly becomes impractical.

Due to the difficulty in constructing optimal search procedures for scheduling problems, several special purpose heuristic methods have been proposed for solving different problems. Two examples for the job shop scheduling problem are Adams et al.'s [1988] shifting bottleneck heuristic and Applegate and Cook [1991] heuristics that utilize ideas from both their

branch and bound algorithm and the shifting bottleneck heuristic. Unfortunately, it is difficult to modify these types of heuristics to consider the complexities found in the automaker data.

Because of the difficulty of using branch-and-bound and special purpose heuristics for many scheduling problems, general heuristic search techniques have been applied to scheduling problems in recent years. Tabu search (Glover [1989] and Glover [1990]) has been applied to scheduling problems by Barnes and Laguna [1993], Dell'Amico and Trubian [1993], Nowicki and Smutnicki [1993], and Taillard [1994]. Both Nowicki and Smutnicki [1993] and Taillard [1994] succeeded in finding new best solutions for several problems that do not yet have known optimal solutions. Simulated annealing (Kirkpatrick et al. [1983] and Černy [1985]) applications to the job shop problem are discussed in Van Laarhoven et al. [1992] and Aarts et al. [1994]. Good results have been found for several of the classic test problems from Lawrence [1984] and Adams et al. [1988]. Genetic algorithms (GA) have also demonstrated potential for solving scheduling problems. A more detailed description of previous GA efforts will be given in Section 2 where our genetic algorithm is discussed.

2 Proposed GA Solution Methodology

We propose a genetic algorithm approach for solving complex scheduling problems. Genetic algorithms (GAs) were introduced by Holland [1975] as a method for modeling complex systems. GAs apply concepts from biological evolution to a mathematical context. The general idea is to start with randomly generated solutions and, implementing a "survival-of-

the-fittest" strategy, evolve good solutions. See Goldberg [1989], Davis [1991], or Michalewicz [1994] for additional information.

GAs have been applied to scheduling problems by several authors. The primary strategy that has been used is a literal permutation ordering encoding (Bagchi et al [1991], Cleveland and Smith [1989], Nakano [1991], Syswerda [1991], Falkenauer and Bouffouix [1991], Whitley, Starkweather and Shaner [1991]). In the literal encoding strategy the chromosome has a one to one correlation with the job sequence. For a five job, single machine problem consider the chromosome

$$3 \rightarrow 1 \rightarrow 5 \rightarrow 4 \rightarrow 2.$$

This represents a sequence in which job 3 is sequenced first, job 1 is sequenced second, and likewise until job 2 is placed fifth in the sequence. This encoding has the advantage of simplicity. The appropriate schedule can be constructed directly from the chromosomal sequence. However, this encoding leads to infeasible job sequences if the traditional one-point crossover operator is used. Applying crossover may result in sequences that contain the same job in two different locations while other jobs are not present in the sequence. To overcome this crossover difficulty several specialized operators have been developed to insure the feasibility of generated solutions. Some of these include PMX Crossover (Goldberg and Lingle [1985]), the subsequence-swap operator (Grefenstette et al. [1985]), the subsequence-chunk operator (Grefenstette [1987]), other subsequence operators (Cleveland and Smith [1989]), edge recombination (Whitley et al. [1989]), order-based and position-based crossover (Syswerda [1989]), linear order crossover Falkenauer and Bouffouix [1991], and forcing (Nakano [1991]). These specialized operators have been found to perform well for some permutation ordering

problems.

Two of the more innovative applications of GAs to scheduling problems are the Problem Space and Heuristic Space methods of Storer, Wu, and Vaccari [1992]. Storer, Wu, and Vaccari [1992] begin from the premise that most scheduling problems (or other problems - but we will focus on the scheduling domain), have base heuristics that are fast and obtain good solutions, though not necessarily optimal ones. Such heuristics may include SPT, EDD, modified due date and more complicated variations. They note that, for difficult scheduling problems, such heuristics do not always find outstanding solutions. They go on to conjecture that if the data were perturbed, the heuristic may find a solution that is outstanding for the original problem. The problem space approach searches this set of perturbations using different search mechanisms including GAs. Heuristic space introduces another level of search. It is possible to conduct heuristic space searches in several different ways. Storer, Wu, and Vaccari [1992] describe two of these methods. In the first method the space has dimension equal to the cardinality of a set of base heuristics (perhaps EDD, SPT and others more complicated). A new heuristic is formed by weighting the elements of the heuristic space. These weights are chosen by a genetic algorithm search over a space of weights. In the second method the scheduling decisions are divided into groups or "windows" and a dispatching rule is assigned to each window. A search is made of different combinations of dispatching rules for the set of windows. Because the allele values represent perturbed problem data or heuristic rules, no infeasibilities are introduced by the crossover operator. Note that the heuristic search space can also be searched using several different search techniques. These methods have been tested on several of the classical job shop test problems

(Storer, Wu, and Park [1992]) and found to perform well.

We utilize an encoding that is different than those typically used with GAs. The random keys representation (Bean [1994]) encodes a solution with *random* numbers. These values are used as sort *keys* to decode the solution. Chromosomal encodings are constructed that represent solutions in a soft manner. These encodings are interpreted in the fitness evaluation routine in a way that avoids the feasibility problem. A similar idea was presented by Shaefer and Smith [1988]. They embedded their encoding in a genetic algorithm that dynamically adjusts its internal representation of the search space according to the problem being solved. There are other encodings that use random variates (e.g. Bäck, Hoffmeister, and Schwefel [1991]) but not in the manner of random keys. Bagchi et al. [1991] explore different scheduling problem encodings but none that are similar to random keys. A strength of random keys is its robustness. The approach works for many other problems. Bean [1994] describes the use for scheduling, vehicle routing, resource allocation, and quadratic assignment problems. Bean [1994] shows excellent computational results for identical machine scheduling and resource allocation and reasonable computational results for the quadratic assignment problem.

An example of the random keys encoding is now provided for the context of scheduling. As a foundation, consider the single machine sequencing problem. Begin by generating a uniform $(0, 1)$ random variate for each job. When such a sequence of realizations is passed to the fitness evaluation routine, sort them and sequence the jobs in ascending order of the sort. For a five job single machine problem, consider the chromosome

$$(.29, .96, .17, .84, .48).$$

Sorting the keys results in the sequence

$$3 \rightarrow 1 \rightarrow 5 \rightarrow 4 \rightarrow 2.$$

If our objective is to minimize total tardiness, this sequence can be evaluated by calculating and summing tardiness. Note that many random key vectors would sort to the same sequence.

The random keys GA (RKGA) operates in two spaces, the chromosome space and the schedule assignment space. As in Storer, Wu, and Vaccari [1992], the use of two search spaces facilitates the search for good problem solutions. For a five job single machine problem, the chromosome space consists of all points within a 5-dimensional hypercube, $\mathcal{C} = [0,1]^5$. Define chromosome $x \in \mathcal{C}$ where $x = \{x_1, x_2, ..., x_5\}$. Let x_i represent the random key value for allele i of chromosome x. The schedule assignment space, $\mathcal{S}$, contains all the possible schedules for the problem. For a single machine problem with regular measure, this consists of all the possible permutations of the jobs. For the single machine problem, $\mathcal{C}$ is mapped to $\mathcal{S}$ by sequencing the jobs based on the sorted random key values.

The random keys encoding has the advantage, over a literal encoding, that all crossovers produce feasible sequences. Crossovers are executed on the chromosomes, the random keys, not on the sequences. Therefore the offspring always contain random keys that can be sorted into an ordered set. Since any ordered set of random keys can be interpreted as a job sequence, all offspring are feasible solutions. The random keys simply serve as tags that the crossover operator uses to rearrange jobs.

The random keys encoding works through the dynamic of the GA. Jobs that should be

early in the sequence evolve low random key values and jobs that should be late in the sequence evolve large random key values. Recall the parent and offspring chromosomes from the previous example. If job 2 should be early in the sequence, chromosomes containing the random key .14 for gene 2, as in parent 2 and offspring 2, will tend to have better fitness values than chromosomes with larger random key values for gene 2. In successive generations the chromosomes with a large random key value for gene 2 will be selected less frequently. Gene 2 will evolve small random key values in subsequent generations.

The RKGA has some constructs that are similar to other GA methods in the literature - most notably the Problems Space method of Storer, Wu, and Vaccari [1992]. However, these two methods differ in several ways. There is a significant difference in the search philosophy. The random keys approach does not use scheduling heuristics while use of scheduling heuristics is "fundamental" (Storer, Wu, and Park [1992]) to the Problem Space method. There is also a difference in the search spaces investigated by the Problem Space method and the RKGA. As problems get more complex and the base heuristic requires more information, the size of an element of the problem space varies. The random keys space always has dimension equal to the number of jobs. As problem complexity increases (add release times, add due times, add tool constraints, add sequence dependent setups), the space does not change. It is always homeomorphic to $[0,1]^n$. The two methods also construct schedules in a different manner. The key difference is that the method in our paper naturally constructs inserted idle time where appropriate. Storer, Wu, and Vaccari [1992] employ a delay factor to allow inserted idle time.

3 GA Encoding for Complex Scheduling Problems

The RKGA can be applied to scheduling problems with multiple, nonidentical machines, nonzero ready times, sequence dependent setups, tool constraints, and precedence. As the problem complexity increases the mapping from the chromosome space to the schedule assignment space becomes more complicated. Demonstrating that the algorithm converges requires proof that there is a region in the chromosome space, $\mathcal{C}$, with nonzero volume that maps to the set of optimal schedules. Additionally, the mapping from chromosomes to schedule assignments provides an opportunity to utilize problem specific data to enhance the performance of the RKGA. Each of these issues are explored in more detail below.

3.1 Modifying the GA for Additional Problem Complexities

Complications such as those mentioned above occur in real scheduling environments. In general, not all of the jobs are available at the beginning of the study horizon. They arrive throughout the study horizon. Limited numbers of tools may restrict concurrent processing of some jobs. Sequence dependent setups are a natural result of the varying tool requirements of the jobs. These complications make the problem more difficult because optimal schedules often contain unforced idle time. In addition, good solutions may require sequencing that sacrifices local evaluations to obtain a better global evaluation. We now discuss in more detail the enhancements to the GA necessary to solve these more difficult problems. Changes are required in the encoding and in the function evaluation routine, in particular, in the schedule construction step.

The random keys approach can be extended to multiple machine problems by extending the encoding. As an intermediate step, consider the m identical machine n job problem to minimize total tardiness. For each job generate an integer randomly in $\{1,\ldots,m\}$ and add a uniform $(0,1)$ variate. The integer part of any random key is interpreted as the machine assignment for that job. Sorting the fractional parts provides the job sequence on each machine. Assuming that jobs are processed at their earliest possible time, a schedule can be constructed and evaluated for total tardiness. The schedule can contain unforced idle time by preserving the sorted order of the job sequence. If the machines are not identical, generate the integer randomly from the set M_i where M_i contains all machines that can be used to process job i.

Nonzero ready times are incorporated into the GA in the schedule construction phase. After the random keys are sorted, the job sequence and machine assignments for each job have been determined. A schedule can be constructed by starting the next job in the sequence at the maximum of its ready time and the time its assigned machine is next available.

Sequence dependent setups are readily incorporated into the schedule. Variables are updated that indicate the last job performed on each of the machines. As the schedule is constructed from the job sequence and machine assignments it is a simple table lookup to determine the setup time resulting from that job assignment.

Tool constraints are captured by updating a tool availability indicator variable. If a job requires a given tool then it may not start until that tool is available. The tool is then marked unavailable for the duration of the job's processing.

Nonzero ready times, sequence dependent setups, and tool availability can all be considered simultaneously by requiring that a job's start time be the maximum of its ready time, completion time for the job's predecessor plus the required setup time, and the tool available time.

The mechanisms for incorporating nonzero ready times and tool constraints into the schedule may appear inefficient because they may introduce idle time into the schedule. However, the ability to create schedules that contain unforced idle time is one of the principle advantages of using a job sequence as the foundation for schedule construction. It is not difficult to construct example problems where inserting idle time into the schedule actually improves the overall objective.

Precedence constraints can also be captured by the RKGA. Recall that the sorted random keys provide a job sequence. Jobs are scheduled using this sequence subject to the constraint that a job cannot be scheduled unless its predecessor(s) have been scheduled. A pseudocode for handling precedence constraints is presented below. First we define additional terminology and notation. A schedule contains a *local left-shift* if it is possible to move a job to the left on the Gantt chart while preserving the job sequence (Baker [1974]). A schedule is *semi-active* if it contains no local left-shifts (Baker [1974]). Let the set J represent the set of jobs remaining to be scheduled ordered by random keys. Let the set K represent the set of schedulable jobs based on precedence. Construct the ordered subset $J \cap K$.

Pseudocode for Handling Precedence Constraints

1. Initialize J and K.

2. Select the first element of $J \cap K$, j.

3. Schedule j on its assigned machine so that no local left-shift is possible.

4. Update J and K by removing j from both sets. Add any jobs to K that became scheduleable because j was scheduled.

5. If $J \neq \{\emptyset\}$ return to step 2.

Precedence can be combined with nonzero ready times, sequence dependent setup times, and tool constraints. The precedence constraints are used to produce the sequence that is then scheduled taking into account the other factors, such as nonzero ready times, sequence dependent setups and tooling.

The mechanisms for incorporating nonzero ready times, sequence dependent setups, tool constraints, and precedence are computationally simple. This is important because the GA must perform numerous function evaluations.

3.2 Mapping from $\mathcal{C}$ to $\mathcal{S}$

As the problem complexity increases, the schedule construction mapping becomes more complicated. It is essential that the random keys encoding ensure that there exists a region in $\mathcal{C}$, with nonzero volume, that maps to the optimal schedule set in $\mathcal{S}$. Then a search for this region in $\mathcal{C}$ can be used as a surrogate for a search of $\mathcal{S}$ for an optimal schedule. Constructing schedules in the manner previously described, Theorem 1 shows that such a region exists within $\mathcal{C}$ for problems with a regular measure. Note that we also assume that

there is no job preemption.

The proof of Theorem 1 and supporting proofs use the following notation:

T_i	The tool requirement for job i.
S_i	The start time of job i.
MA_j	The time when machine j is next available for processing.
TA_j	The time when tool j is next available for use.
LT_k	The last tool used on machine k.
MTL_j	The last machine on which tool j was used.
α_i	The random key value for job i, $\alpha_i \in [0,1]$ for all $i = 1,2,\ldots,n$.
M_i	The set of machines that can perform job i.
$[i]$	Order statistic notation for the jobs from a sort of the α_i values.
d	The total number of tools in the problem.
SA	The set of semi-active schedules.

Remark: T_i is assumed scalar in the following proofs but is readily generalizable to a set.

The RKGA uses the procedure $\gamma(x)$ to map from $\mathcal{C}$ to $\mathcal{S}$. The procedure $\gamma(x)$ is described below.

Let $x \in \mathcal{C}$ have $x_i = (m_i, \alpha_i)$ where $m_i \in M_i, \alpha_i \in [0,1]$ for all $i = 1,2,\ldots,n$.

Procedure $\gamma(x)$

Sort $\{\alpha_i\}_{i=1}^n$ to obtain $[1],[2],\ldots,[n]$.

Set $MA_k = 0, LT_k = \emptyset \ \forall k = 1,2,\ldots,m$.

Set $TA_h = 0 \ \forall h = 1, 2, \ldots, d.$

For$(i = 1; i \leq n; i++)$

{

if $T_{[i]} = LT_{m_{[i]}}$ and $MTL_{T_{[i]}} = m_{[i]}$

Setuptime=0

else Setuptime=Setupvalue

$$S_{[i]} = max\{max\{MA_{m_{[i]}}, TA_{T_{[i]}}\} + Setuptime, r_{[i]}\} \tag{1}$$

$MA_{m_{[i]}} = TA_{T_{[i]}} = S_{[i]} + p_{[i]}$

$LT_{m_{[i]}} = T_{[i]}$

$MTL_{T_{[i]}} = m_{[i]}$

}

The outcome of γ is a set of start times, S_i, that, together with the machine assignments from the random keys, describe a schedule.

Lemma 1 *For all $x \in \mathcal{C}, \gamma(x)$ is a feasible schedule.*

Proof:

By construction, no jobs on the same machine are processed simultaneously. The same is true for jobs that require the same tool. Setup time is accounted for and no job may begin prior to its ready time. Hence the schedule is feasible. ∎

In Lemma 2 we show that the image of $\mathcal{C}$ under γ is exactly the set of semi-active schedules.

Lemma 2 $\gamma(\mathcal{C}) = SA$.

Proof:

$\gamma(x)$ is semi-active for all $x \in \mathcal{C}$.

If not, there must exist a local left-shift. Some job, i, could be shifted left without violating machine, tool, or ready time constraints. This contradicts Step (1) of procedure $\gamma(x)$. Hence $\gamma(x)$ is semi-active.

For each semi-active schedule, $s \in SA, \exists x \in \mathcal{C} \ni \gamma(x) = s$.

Note that each semi-active schedule s can be represented as a set of ordered pairs of machine assignments and start times $\{(m_i, S_i)\}_{i=1}^{n}$. Let $\overline{S} = \max_{i=1,2,\ldots,n} S_i$. Then $\hat{x} \equiv \{(m_i, \frac{S_i}{\overline{S}})\}_{i=1}^{n} \in \mathcal{C}$.

We claim that $\gamma(\hat{x}) = s$. All jobs are on the same machine in s and $\gamma(\hat{x})$. The jobs are placed in the same order since $\frac{S_i}{\overline{S}}$ is ordered the same as S_i. Since MA and TA are monotone increasing during construction of $\gamma(x)$, jobs that require the same tool or are assigned to the same machine are processed in the same order as they are placed. Setups are the same since the job sequence for each machine and each tool is the same. Since the machine assignments, setup times, and job sequence on each machine are the same for $\gamma(\hat{x})$ and s then $\{S_1, S_2, \ldots, S_n\}$ are the same for s and for $\gamma(\hat{x})$. ∎

In Theorem 1 we prove that the GA finds an optimal solution with probability 1. Note that the algorithm presented in Section 5 generates a random sample at least once per generation. This random sample is defined as an immigrant. Let x^t be the best solution found after t generations. $Y \subset \mathcal{S}$ is the set of optimal schedules.

Theorem 1 $\lim_{t\to\infty} P(\gamma(x^t) \in Y) = 1$.

Proof:

For a regular measure, Baker [1974] proved $(Y \cap SA) \neq \emptyset$. By Lemma 2, $\gamma^{-1}(Y \cap SA) \neq \emptyset$. Let $x \in \gamma^{-1}(Y \cap SA)$. Then $x = \{(m_i, \alpha_i)\}_{i=1}^n$ where $m_i \in M_i$ and α_i is a random variate from a uniform $(0,1)$. Let $\Delta = \min\{\min_{i,j} |\alpha_i - \alpha_j|, \min_i(\alpha_i, 1-\alpha_i)\}$. Since there are a finite number of jobs and the α_i are generated from a uniform (0,1), $\Delta > 0$ almost surely. Define the neighborhood of $x, N(x) = \{x' : m_i' = m_i, |\alpha_i' - \alpha_i| < \frac{\Delta}{2} \ \forall i = 1, 2, \ldots, n\}$. If $x' \in N(x)$ then $\gamma(x') = \gamma(x) \in Y$. When randomly generating $x' \in \mathcal{C}$,

$$P(x' \in N(x)) \geq (\prod_{i=1}^{n} \frac{1}{|M_i|})\Delta^n = \varepsilon > 0.$$

Now, in each generation, the immigrant, x, which is uniformly generated from $\mathcal{C}$, has probability $\geq \varepsilon$ of being optimal. The result follows immediately. ∎

This result is of value as it establishes the asymptotic convergence of the algorithm. It is unsatisfying in that it relies on immigration rather than crossover to drive the theoretical convergence. Empirically, crossover is much more important. Understanding the theoretical rate of convergence, based on crossover, is an open research question.

As problem complexities are added, the schedule assignment space, $\mathcal{S}$, becomes increasingly complicated. Each point in this space must, essentially, be a Gantt chart of the schedule including: job to machine assignments, the job sequence for each machine, and possible interactions between jobs on different machines. These interactions may include precedence constraints and competition for scarce resources such as tooling. Due to the complexity of $\mathcal{S}$, it is impractical to search it directly. In the RKGA, all operators are defined on $\mathcal{C}$ which

is stable. The mapping γ is altered to absorb these complexities.

The robust nature of the GA is demonstrated by the fact that the same algorithm can solve different scheduling problems by making changes only to the encoding and the fitness evaluation function. The mechanisms of reproduction and mutation are unaffected.

3.3 Enhancing GA Performance Using Problem Specific Data

The scheduling application incorporates problem specific information into the random keys encoding to improve the rate of convergence. Recall that for the general random keys encoding the random keys for all of the genes are uniform $(0, 1)$ variates. The scheduling application contains problem specific data which can be used to *bias* the random key values of the jobs. If the problem objective is to minimize total tardiness then it is likely that jobs that have early ready and due times will be found early in the optimal sequence. Likewise, jobs with late ready and due times will probably be found late in the optimal sequence. We bias the generation of alleles in $\mathcal{C}$ to make jobs with early ready and due times have a higher probability of being placed early in the sequence. Consider two jobs

$$\text{Job 1} \quad r_1 = 10 \quad d_1 = 25$$

$$\text{Job 2} \quad r_1 = 30 \quad d_1 = 65$$

Assume the random key values for jobs 1 and 2 are selected using biased sampling and that the mean for job 1 is lower than the mean for job 2. On average job 2 receives a larger random key value than job 1 and therefore is later in the sequence than job 1. However, it is

still possible for job 2 to precede job 1 (assuming that there is no precedence conflict). This is important because the optimal schedule may have job 2 preceding job 1 due to tooling constraints, sequence dependent setups or other factors. The entire solution space can still be explored but the biasing pushes the search in a direction where the optimal is most likely to be found.

Choice of factors to use for biasing, and the relative weights of those factors, are open questions. If the bias is too small, then the effect is lost and the algorithm returns to the general random key method. If the bias is too large, then the algorithm converges prematurely. In this study all biasing came from ready times and due times. These were selected because ready time represents a hard constraint - a job cannot begin prior to its ready time - and due time is directly related to the tardiness based objective function.

4 GA Operators

There are several types of genetic operators that may be used in the reproduction stage of a GA (see Goldberg [1989]). Because the random keys encoding preserves feasibility there is no need to use the specialized operators discussed in Section 2. The code described here uses elitist reproduction, Bernoulli crossover, immigration, and post-tournament selection to fill the next generation. The random keys representation is not limited to these operators. Other operators remain to be investigated. However, these operators have performed well in tests completed at this time.

Elitist reproduction (Goldberg [1989]) is accomplished by copying the best individuals

from one generation to the next. This method has the advantage of maintaining a best solution that is monotonically improving. However, it has the potential for premature convergence. This is overcome by introducing high mutation rates.

Bernoulli crossover (called parameterized uniform crossover in Spears and De Jong [1991]) is used in place of the traditional single-point or multiple-point crossover. Two chromosomes are selected randomly with replacement from the current population to function as "parents." Let $X = (x_1, x_2, ...x_l)$ and $Y = (y_1, y_2, ...y_l)$ represent the l random key alleles in these two chromosomes. Let $Z = (z_1, z_2, ...z_l)$ represent l independent uniform $(0, 1)$ variates and P_c represent the probability of crossover for each gene. Let $W = (w_1, w_2, ...w_l)$ and $V = (v_1, v_2, ...v_l)$ be the l random key alleles for the two offspring that will result from the crossover of the two parent chromosomes. Determine each allele in W and V as follows:

$$\begin{cases} w_i = x_i \text{ and } v_i = y_i & \text{if } z_i < P_c \\ w_i = y_i \text{ and } v_i = x_i & \text{if } z_i \geq P_c \end{cases}$$

An example is given in Table 1.

Table 1: Bernoulli crossover example with $P_c = 0.7$

Random Input Z	0.64	0.15	0.76	0.33	0.91
Parent X	.29	.96	.17	.84	.48
Parent Y	.73	.14	.43	.54	.32
Offspring W	.29	.96	.43	.84	.32
Offspring V	.73	.14	.17	.54	.48

Implementing a mutation operator for a real coded GA presents difficulties. For a binary encoding, mutation consists of flipping one bit of the gene with a given probability. It is not obvious how to translate bit flipping in a binary encoding to a real valued encoding. Janikow and Michalewicz [1991] discuss possible mechanisms for implementing single gene mutation for real valued encodings. We avoid these difficulties by using immigration, rather than traditional single gene mutation, to replace lost genetic material. The immigration operator creates a new randomly generated individual in each next generation. This individual may then be selected as a parent for the crossover operation in succeeding generations. This introduces a significant mutation effect which offsets the premature convergence effect of elitist reproduction and tournament-type selection.

A type of tournament selection, which we label post-tournament selection, is used in conjunction with the crossover operation to fill the next generation. Two chromosomes are chosen randomly with replacement from the entire previous generation to serve as parents for the crossover operation. Performing crossover on these two chromosomes, X and Y, results in the two offspring V and W. Both of these chromosomes are evaluated and only that with better fitness is permitted to enter the pool of potential parents for the next generation. This procedure is repeated until a new generation of chromosomes is created. Post-tournament selection is similar to tournament style selection with $s = 2$ as described in Brindle [1981]. However, the convergence pressure is stronger with post-tournament selection because half of the offspring are not even permitted to join the pool of potential parents.

The definition of population convergence for the RKGA varies from that common in the GA literature. In the traditional definition of population convergence, all chromosomes in

the population are identical (Goldberg [1989], Davis [1991]). We do not use this definition of convergence for three reasons. First, the behavior of the immigration operator makes it very improbable that all members of the population will ever be the same. Second, because different chromosomes of random keys can map to the same schedule assignment, the population can converge to the same schedule assignment without converging in the chromosome space. Third, empirical results indicate that if the GA does not improve the current best solution for 15 successive generations then additional searching yields little improvement. For this study, population convergence is assumed if the best solution does not change for 15 generations.

Having discussed the random keys encoding and the operators used in the RKGA, a pseudocode for the algorithm is presented below. Step 2 is repeated for each generation until either the population converges or until the maximum number of generations is reached. Step 2.2 implements the elitist strategy and copies the *number_of_clones* best members from the entire previous generation. Step 2.3 represents the reproduction step of the algorithm. The crossover operation is performed using two parents that are randomly selected from the previous generation. This step is repeated to create all members of the new population that are not copied over directly from the previous generation. Then the population is ranked based on fitness and the *number_of_immigrants* worst members of the population are replaced with immigrants in order to enhance diversity in the gene pool.

Pseudocode for the Random Keys GA

1. Initialize population. Rank the population based on fitness. Set stop=0, count=0.
2. While stop=0
   - 2.1 Count=Count+1
   - 2.2 Copy the *number_of_clones* best members from the previous generation to the new generation
   - 2.3 For i=1 to *population_size* − *number_of_clones*
     - 2.3.1 Randomly select two parents from the previous generation.
     - 2.3.2 Perform the Bernoulli crossover operation.
     - 2.3.3 Evaluate the fitness of the two offspring.
     - 2.3.4 Keep the better of the two offspring and place it in the next generation. Discard the other offspring.
   - 2.4 Rank the population based on fitness.
   - 2.5 Replace the *number_of_immigrants* worst members of the population with imr
   - 2.6 If the population has converged set stop=1.
   - 2.7 If count=*maximum_number_of_generations* set stop=1.
3. Output the final solution.

5 Computational Results

To determine the effectiveness of the RKGA for scheduling, several problem sets were tested. The first two data sets represent actual problems from an automaker. The third and fourth data sets are from the literature.

5.1 Real Auto Problems

The first and second data sets comprise a total of twelve problems from a large automaker (Bean et al. [1991]). The seven problems in the first data set each contain approximately 360 jobs that are to be scheduled on two machines. Recall from Section 1 that these problems contain the following complexities: nonzero ready times, due dates, tooling constraints, sequence dependent setup times, machine flexibility, and total tardiness objective. The seven problems are derived from a single data set and represent seven disruptions of that data set. The first problem represents the undisrupted case. The remaining six problems contain data disruptions including: periods of machine downtime, periods of tool unavailability, and changes in p_i, r_i, and d_i. The presence of disruptions significantly changes the data set and creates a different scheduling problem.

The second data set contains five disruptions of a data set containing approximately 270 jobs to be scheduled on ten non-identical machines. The additional complexities are the same as those for the first data set. They are captured from a different auto plant than data set 1.

Due to the large size and complexity of the problems in the first and second data sets, it was not possible to compute optimal solutions for these problems. Therefore, the RKGA's solutions are compared to total tardiness lower bounds for each of the problems. The lower bound for each problem is the inherent tardiness for the problem.

$$LB = \sum_{i=1}^{n} max(0, r_i + p_i - d_i)$$

The RKGA's results are also compared to the Matchup scheduling methods of Bean et al.

[1991] for data set 1.

The GA was run ten times for each problem using a different random seed for each run. The code was implemented on a Maspar MP-1 massively parallel computer with 1024 processors. For each performance criterion, the minimum, median, and maximum values for the ten random seeds are presented. The minimum and maximum values provide best and worst case performance measures. The median is used rather than the mean to present the average performance in order to reduce the effect of extreme minimum and maximum values.

Table 2 compares the lower bound for each of the twelve problems in the first and second data sets with the best solution obtained using the GA. The results indicate that solutions that are within 0.3 to 6.3% of the lower bound can be found in about 5 to 9 minutes of cpu time for the seven disruptions of data set one. Across the seven problems, the median solution is 0.7% to 5.8% above the lower bound and requires 6 to 8 minutes of cpu time. The five problems in the second data set require 3 to 7 minutes of computation time. For these problems the percent deviation from the lower bound is not meaningful because the lower bound is zero. However, the maximum total tardiness was less than 18 units for all seeds tested across all the problems in the second data set. Table 2 also provides a comparison between the RKGA and the Matchup Scheduling Algorithm (MUSA) of Bean et al. [1991] for data set one. These results indicate that the RKGA consistently finds better solutions than the MUSA. Across the seven problems the median (across the 10 seeds) RKGA solution is an average of 8% better than the MUSA solution. Both the RKGA and the MUSA exhibit their worst performance on Problem 1.1. This seems to indicate that this problem is more difficult than the others and/or that the lower bound is not as tight for this problem.

Table 2: Automaker Data Test Results. Each line represents 10 runs with different random seeds.

	RKGA									MUSA
	Number of Generations			CPU Time†			Percent Above Lower Bound			Percent Above Lower Bound
Problem	Min	Median	Max	Min	Median	Max	Min	Median	Max	
1.1	156	187	208	393	462	511	5.1	5.8	6.3	25.4
1.2	140	160	181	356	401	449	0.8	1.1	1.4	5.6
1.3	136	161	200	354	404	492	1.0	1.3	1.5	7.7
1.4	140	162	206	356	405	507	0.4	0.8	1.2	6.2
1.5	142	160	183	360	401	454	0.3	0.7	1.6	8.5
1.6	122	143	164	314	363	410	0.4	0.9	1.1	5.1
1.7	152	173	194	383	432	479	1.3	1.5	1.7	6.3
2.1	119	140	187	208	271	353	0.0*	N/A	N/A	-
2.2	108	119	168	189	227	320	0.0*	N/A	N/A	-
2.3	111	132	150	195	243	288	0.0*	0.0*	N/A	-
2.4	149	192	218	287	362	407	0.0*	N/A	N/A	-
2.5	106	143	174	186	276	331	0.0*	0.0*	N/A	-

†Seconds on Maspar MP-1

N/A - Not Applicable. Percent above lower bound is not meaningful because the lower bound for these problems is zero.

* Attained lower bound of zero.

- Same data set not run with MUSA.

Test results for the problems of data set one show that the GA can find solutions within 5% of the lower bound in about one half the time required to converge to a final solution. This demonstrates how the GA quickly finds good solutions and then spends additional time to obtain the final improvements. This is a useful property for real scheduling implementations since the algorithm can be terminated sooner with only a small reduction in performance. The time to find solutions within 5% of the lower bound is not meaningful for the five problems of the second data set because the lower bound is 0.

As discussed in Section 1, with the exception of Bean et al. [1991], we have not found problems of this size and complexity solved in the literature. A branch-and-bound method could be developed to solve the problem. This does not represent a practical solution approach because branch-and-bound has long and variable run times on problems of the size and complexity of those analyzed in this paper. Therefore we have compared the performance of the GA with ad hoc solution methods. Comparisons with the dynamic priority rules tested in Bean et al. [1991] indicate that the RKGA finds solutions with 18% less tardiness on average.

Additional heuristics that utilize list processing rules such as Earliest Due Date(EDD, Jackson [1955]), Modified Due Date(MDD, Baker and Kanet [1983]), or Earliest Release Date (ERD) were also tested. These list processing rules provide a job sequence but do not provide job to machine assignments. A GA was used to determine good job to machine assignments given a job sequence that resulted from a list processing rule. The median percent above the lower bound ranged from 23 to 30% for EDD and 21 to 28% for MDD for the 7 problems in the first data set. These values are significantly larger than the results for

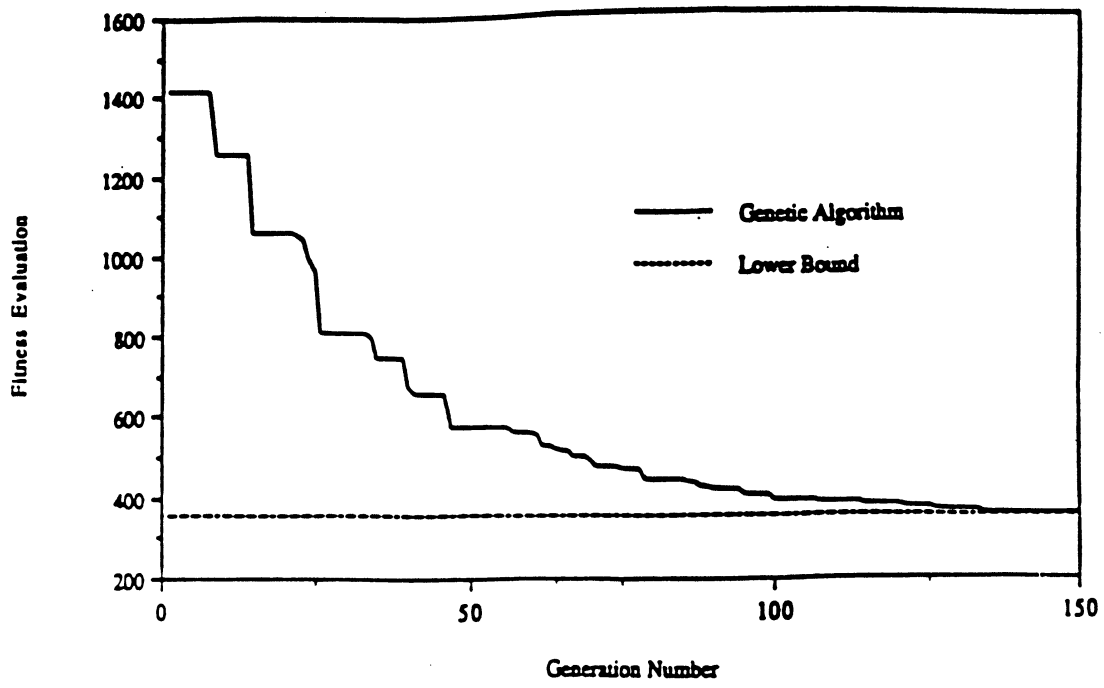
the RKGA given in Table 2. The RKGA finds much better schedule assignments than either EDD or MDD based schedules and requires 10% and 70% less computation time for the first and second data sets respectively. ERD performs worse than EDD or MDD. Compared with the RKGA, ERD finds solutions that have a fitness evaluation that is approximately twice as much for data set one and about 10 times as large for data set two. The computation time for the RKGA is 33% less than the time for ERD for data set one and is 66% less than the time for ERD for data set two. Thus, the performance of the RKGA dominates ERD.

Figure 1 provides the GA's convergence graph for a random seed of data set 1.7. The graph indicates the smooth pattern of convergence for the GA. Convergence graphs of the GA for the other disruptions in the first and second data sets are similar to Figure 1. The GA explores many schedule assignments but these represent a very small fraction ($\ll$.01%) of the entire schedule assignment space. The strength of the GA is the ability to combine segments of different schedule assignments to construct a good schedule assignment for the entire problem.

5.2 Problems from the Literature

The robust nature of the RKGA was verified by testing two additional problem types from the literature. The first, from Hoitomt et al. [1993], contains jobs that contain release times, due times, alternative machine assignments, and can contain multiple operations. The objective is to minimize the sum of the tardiness squared. The second problem type, from Rubin and Ragatz [1995], are single machine problems that contain sequence dependent setup times and due dates. The objective is to minimize the sum of the total tardiness. The combination

Figure 1: GA Convergence.



of sequence dependent setup times and tardiness constitutes a difficult problem even for the single machine case. The same algorithm that was run on the automaker data was run on these problems with slight modifications to the schedule construction and evaluation routines to account for changes in problem structure and objective function. The parameter settings were held constant.

The data from Hoitomt et al. [1993] has jobs that contain from 1 to 6 operations and each job has a release date and a due date. Each of the operations within a job can be performed on from 1 to 6 machines. The objective is to minimize the weighted sum of the tardiness squared. The particular problem instance tested contains 127 jobs and a total of 187 operations. This problem was selected because it contains many of the complexities also

found in the automaker data. The RKGA was applied to this problem using 10 different seeds. For every seed the RKGA found solutions within 1% of the lower bound. The run times for the different seeds ranged from 4 to 6 minutes. The time to find solutions within 5% of the lower bound ranged from 15 to 60 seconds.

Rubin and Ragatz [1995] generated 32 single machine problems that have sequence dependent setups and an objective of minimizing total tardiness. The 32 problems comprise four problem sizes: 15, 25, 35, and 45 jobs. For each problem size there are eight problem instances that were generated by varying 3 problem factors. The first factor is processing time variance which can be either high or low. The second factor is the tardiness factor which can be low or moderate. The tardiness factor roughly corresponds to the number of jobs that will be tardy in a randomly generated sequence. The third factor concerns the due date factor which can be either narrow or wide. Exploring all combinations of each factor level results in eight problems for each problem size.

Rubin and Ragatz [1995] investigated three solution methods for these problems: branch-and-bound, a GA based heuristic, and an interchange heuristic. Due to long computation times, the branch-and-bound could only verify the optimality of the 15 job problems and some of the larger problems that attained a total tardiness of zero. For the remaining problems the branch-and-bound search was stopped after exploring a maximum of two million nodes. The first heuristic methodology, referred to as R&RGA, utilizes a GA in conjunction with an interchange heuristic. The interchange heuristic explores all possible pairwise exchanges of jobs in the sequence and accepts all moves that improve the objective. This interchange heuristic is applied to the best solution found at each generation and a randomly selected

fraction of the remainder of the population. The second approach, referred to as R&RIH, applies the interchange heuristic to a randomly generated solutions. Both heuristic searches were terminated once a solution was found that matched or beat the best solution found by the branch-and-bound algorithm.

Test results comparing the RKGA with Rubin and Ragatz's methods are presented in Table 3. Testing for the RKGA was performed by testing 20 random seeds for each problem. The results for the 20 seeds were averaged in order to compare with the results of Rubin and Ragatz [1995] who present the average of 20 heuristic replications. The results for the problems with 25 and 35 jobs reflect the average performance over only 7 of the 8 problems. One problem was deleted for these two problem sizes because the the best incumbent solution found by the branch-and-bound had an objective function value that was significantly more than the best known solution. Given the stopping rule for the heuristics, these two problems do not prove to be meaningful test problems.

Table 3: RKGA Results for Tardiness Problems with Sequence Dependent Setup Times.

Problem Size	RKGA % above optimal	RKGA % below R & R GA	RKGA % below R & R IH
15	1.1	0.5	-0.4
25	-	0.5	0.4
35	-	2.6	1.6
45	-	3.2	3.2

- Optimal solution not known

The results of Table 3 indicate that the RKGA performs well on all of the problems. For the problems with provably optimal solutions the RKGA found solutions that are 1.1% above the optimal. This average falls to 0.15% if one problem that had low tardiness is excluded. (Due to the small tardiness value any deviation from minimum tardiness resulted in a large percent deviation from the optimum value.) The RKGA compares favorably with the other heuristics with the exception of R&RIH for the 15 job problems. The comparison becomes more favorable as the problem size increases.

6 Summary and Conclusions

We have presented a scheduling algorithm that utilizes the random keys encoding within the context of a GA. This algorithm was originally developed to solve complex scheduling problems that arise in the auto industry. Computational tests indicate that the RKGA performed well on all of the auto plant problems tested. The RKGA produced good results across the different random seed values. The median (over the ten seeds) best solution found was within 1.5% of the lower bound for all of the problems in the first data set except 1.1 (it was 5.8%) and was within 18 units of the tardiness lower bound for the second data set. The maximum deviation from the lower bound over all seeds was less than 6.3% for all seven problems of data set one and 21 tardiness units for all five problems of the second data set. The RKGA consistently found schedules with less total tardiness than those found by the current scheduling procedure in only a few minutes of cpu time. Thus, the RKGA provides a good method for generating schedules for problems with the complexities of data sets 1

and 2. In addition, because the schedules can be generated in minutes, this approach can be used to deal with shop floor disruptions in a real time manner.

The robust nature of the RKGA was also demonstrated. The same algorithm used for the automaker problems yielded high quality solutions when applied to the problem of Hoitomt et al. [1993] containing release times, due dates, precedence, and alternative machine assignments. The algorithm was also applied to the single machine problems of Rubin and Ragatz [1995] that contain sequence dependent set up times and due dates and compared favorably with other methods that have been applied to these problems.

7 Acknowledgment

This research was supported in part by National Science Foundation grants DPM-9018515 and DPM-9308432 to the University of Michigan and by the National Science Foundation Graduate Fellowship Program. The authors would like to thank Rosemary D'Onofrio for helping with the data set preparation.

REFERENCES

Aarts, E. H. L., P. J. M. Van Laarhoven, J. K. Lenstra, and N. L. J. Ulder [1994], "A Computational Study of Local Search Algorithms for Job Shop Scheduling," **ORSA Journal on Computing,** Vol. 6, 118-125.

Adams, J., E. Balas, and D. Zawack [1988], "The shifting bottleneck procedure for job shop scheduling," **Management Science,** Vol. 34, 391-401.

Bäck, T., F. Hoffmeister, and H. Schwefel [1991], "A Survey of Evolution Strategies," **Proceedings of the Fourth International Conference on Genetic Algorithms,** 2-9.

Bagchi, S., S. Uckun, Y. Miyabe, and K. Kawamura [1991], "Exploring Problem-Specific Recombination Operators for Job Shop Scheduling," **Proceedings of the Fourth International Conference on Genetic Algorithms,** 10-17.

Baker, K. [1974], **Introduction to Sequencing and Scheduling,** Wiley.

Baker, K. and J. Kanet [1983], "Job Shop Scheduling With Modified Due Dates," **Journal of Operations Management,** Vol. 4, 11-22.

Barnes, J. W. and M. Laguna [1993], "A Tabu Search Experience in Production Scheduling," **Annals of Operations Research,** Vol. 41, 141-156.

Bean, J. C. [1994], "Genetics and Random Keys for Sequencing and Optimization," **ORSA Journal on Computing,** Vol. 6, 154-160.

Bean, J. C., J. R. Birge, J. Mittenthal, and C. Noon [1991], "Matchup Scheduling with Multiple Resources, Release Dates and Disruptions," **Operations Research**, Vol. 39, 470-483.

Brindle, A. [1981], **"Genetic algorithms for function optimization"** (Doctoral dissertation and Technical Report TR 81-2) Edmonton: University of Alberta, Department of Computer Science.

Brucker, P., B. Jurisch, and A. Krämer [1994], "The job-shop problem and immediate selection," **Annals of Operations Research,** Vol. 50, 73-114.

Carlier, J. and E. Pinson [1989], "An Algorithm for Solving the Job-Shop Problem," **Management Science**, Vol. 35, 164-176.

Carlier, J. and E. Pinson [1990], "A practical use of Jackson's preemptive schedule for solving the job shop problem. **Annals of Operations Research**, Vol. 26, 269-287.

Carlier, J. and E. Pinson [1994], "Adjustment of heads and tails for the job-shop problem," **European Journal of Operational Research**, Vol. 35, 164-176.

Černy, V. [1985], "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm," **Journal of Optimization Theory and Applications**, Vol. 45, 41-51.

Cleveland, G.A. and S. F. Smith [1989], "Using Genetic Algorithms to Schedule Flow Shop Releases," **Proceedings of the Third International Conference on Genetic Algorithms**, 160-169.

Davis, L. [1991], **Handbook of Genetic Algorithms**, (ed. L. Davis), Van Nostrand.

Dell'Amico M. and M. Trubian [1993], "Applying Tabu Search to the Job-Shop Scheduling Problem," **Annals of Operations Research**, Vol. 41, 231-252.

Du, J. and J.Y.-T. Leung[1990], "Minimizing Total Tardiness on One Machine is NP-Hard," **Mathematics of Operations Research**, Vol. 15, 483-495.

Falkenauer E. and S. Bouffouix [1991], "A Genetic Algorithm for Job Shop," **Proceedings of the 1991 IEEE International Conference on Robotics and Automation,** 824-829.

French, S. [1982], **Sequencing and Scheduling**, Halstead Press.

Giffler, B. and Thompson, G. L. [1960], "Algorithms for solving production scheduling problems." **Operations Research**, Vol. 8, 487-503.

Glover, F. [1989], "Tabu Search - Part I," **ORSA Journal on Computing**, Vol. 1, 190-206.

Glover, F. [1990], "Tabu Search - Part II," **ORSA Journal on Computing**, Vol. 2, 4-32.

Goldberg, D. E. and R. Lingle Jr. [1985], "Alleles, Loci, and the Traveling Salesman Problem," **Proceedings of the First International Conference on Genetic Algorithms,** 154-159.

Goldberg, D. E. [1989], **Genetic Algorithms in Search Optimization and Machine Learning**, Addison Wesley.

Goldberg, D. E. and K. Deb [1991], "A Comparative Analysis of Selection Schemes Used in Genetic Algorithms," **Foundations of Genetic Algorithms**, Morgan Kaufmann.

Goldberg, D. E. [1991], "Real-coded Genetic Algorithms Virtual Alphabets, and Blocking," **Complex Systems**, Vol. 5, 139-167.

Grabowski, J., E. Nowicki, and S. Zdrzalka [1986], "A Block Approach for Single-Machine Scheduling with Release Dates and Due Dates," **European Journal of Operational Research**, Vol. 26, 278-285.

Grefenstette, J. J., R. Gopal, B. Rosmaita, and D. Van Gucht [1985], "Genetic Algorithms for the Traveling Salesman Problem," **Proceedings of the First International Conference on Genetic Algorithms.**

Grefenstette, J. J. [1987], "Incorporating Problem Specific Knowledge into Genetic Algorithms," **Genetic Algorithms and Simulated Annealing**, (ed. L. Davis) Morgan Kaufmann.

Hoitomt, D. J., P. B. Luh, and K. R. Pattipati [1993], "A Practical Approach to Job-Shop Scheduling Problems," **IEEE Transactions on Robotics and Automation,** Vol. 9, 1-13.

Holland, J. H. [1975], **Adaptation in Natural and Artificial Systems,** The University of Michigan Press, Ann Arbor.

Jackson, J. R. [1955], "Scheduling a Production Line to Minimize Maximum Tardiness," Research Report 43, Management Sciences Research Program, UCLA.

Janikow, C. Z. and Z. Michalewicz [1991], "An Experimental Comparison of Binary an Floating Point Representations in Genetic Algorithms," **Proceedings of the Fourth International Conference on Genetic Algorithms**, 31-36.

Kirkpatrick, S., C. D. Gelatt, Jr., and M. P. Vecchi [1983], "Optimization by Simulated Annealing," **Science**, Vol. 220, 671-680.

Lawrence, S. [1984], "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques (Supplement)." Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh.

Lee, C. and J. Herrmann [1993], "Decision Support Systems for Dynamic Job Shop Scheduling," **Proceedings of the 1993 NSF Design and Manufacturing Systems Conference,** Charlotte, NC, Vol. 2, 1119-1123.

Michalewicz, Z. [1994], **Genetic Algorithms + Data Structures = Evolution Programs**, Springer-Verlag.

Morton, T. E. and D. W. Pentico[1993], Heuristic Scheduling Systems, John Wiley and Sons.

Nakano, R. [1991], "Conventional Genetic Algorithm for Job Shop Problems," **Proceedings of the Fourth International Conference on Genetic Algorithms**, 474-479.

Nowicki, E. and C. Smutnicki [1993], "A Fast Taboo Search Algorithm for the Job Shop Problem," Preprinty nr. 893, Instytut Cybernetyki, Technicznej, Politechniki Wroclawskiej, Poland.

Pinedo, M. [1995], Scheduling Theory, Algorithms, and Systems, Prentice Hall.

Rubin, P. A. and G. L. Ragatz [1995], "Scheduling in a Sequence Dependent Setup Environment With Genetic Search," **Computers and Operations Research**, Vol. 22, 85-99.

Shaefer, C. G. and S. J. Smith [1988], "The ARGOT Strategy II: Combinatorial Optimizations," Technical Report, Thinking Machine Corporation.

Spears, W. M. and K. A. De Jong [1991], "On the Virtues of Parameterized Uniform Crossover," **Proceedings of the Fourth International Conference on Genetic Algorithms**, 230-236.

Storer, R. H., S. D. Wu, and R. Vaccari [1992], "New Search Spaces for Sequencing Problems With Application to Job Shop Scheduling," **Management Science**, Vol. 38, No. 10, 1495-1509.

Storer, R. H., S. D. Wu, and I. Park [1992], "Genetic Algorithms in Problem Space for Sequencing Problems," **Proceedings of a Joint US-German Conference on Operations Research in Production Planning and Control**, 584-597.

Syswerda, G. [1989], "Schedule Optimization Using Genetic Algorithms," in **Handbook of Genetic Algorithms**, (ed. L. Davis), Van Nostrand, 332-349.

Syswerda, G. [1991], "The Application of Genetic Algorithms to Resource Scheduling," **Proceedings of the Fourth International Conference on Genetic Algorithms**, 502-508.

Taillard, E. [1994], "Parallel Tabu Search Techniques," **ORSA Journal on Computing**, Vol. 6, 108-117.

Van Laarhoven, P. J. M., Aarts, E. H. L., and J. K. Lenstra [1992], "Job Shop Scheduling by Simulated Annealing", **Operations Research,** Vol. 40, 113-125.

Whitley, D., T. Starkweather, and D. Fuquay [1989], "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator," **Proceedings of the Third International Conference on Genetic Algorithms**, 133-140.

Whitley, D., T. Starkweather, and D Shaner [1991], "The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination," in **Handbook of Genetic Algorithms**, (ed. L. Davis), Van Nostrand, 350-372.

Widmer, M. [1991], "Job Shop Scheduling with Tooling Constraints: a Tabu Search Approach," **Journal of the Operational Research Society**, Vol. 42, 75-82.

Wright, A. [1991], "Genetic Algorithms for Real Parameter Optimization," **Foundations of Genetic Algorithms**, Morgan Kaufmann.