# A Genetic Algorithm Methodology for Complex Scheduling Problems

**Bryan A. Norman,[1] James C. Bean[2]**
[1] *Department of Industrial Engineering, University of Pittsburgh, Pittsburgh, Pennsylvania, 15261, USA*

[2] *Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan, 48109, USA*

Received July 1996; revised August 1998; accepted 20 August 1998

**Abstract:** This paper considers the 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. © 1999 John Wiley & Sons, Inc. Naval Research Logistics 46: 199–211, 1999

**Keywords:** scheduling; genetic algorithm; heuristic

## 1. INTRODUCTION

We present a random keys genetic algorithm to solve complex scheduling problems. For a comprehensive discussion of scheduling problems and terminology, see [5] and [32]. The principle problem we investigate arises from an automaker. There are several factors that complicate the automaker scheduling problems. The jobs have ready times and due times 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 NP-hard since 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 [15].

*Correspondence to:* B. A. Norman
Contract grant sponsor: National Science Foundation; contract grant numbers: DPM-9018515, DPM-9308432

 CCC 0894-069X/99/020199-13

The vast majority of the papers in the scheduling literature consider only a subset of these problem complexities. In [22], [20], and [42] problems are investigated that contain many of the complexities, but each of these assumes away two or more of the factors. The only paper of which we are aware that has addressed problems containing all the previously mentioned complexities is the Matchup Scheduling work of [9]. They propose different priority rules and heuristic methods based on an integer programming formulation of the problem. Section 3 provides a comparison of the Matchup algorithm with the proposed genetic algorithm.

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 ([16] and [17]) has been applied to scheduling problems by [7], [14], [31], and [39]. Both [31] and [39] succeeded in finding new best solutions for several problems that do not yet have known optimal solutions. Simulated annealing [25, 11] applications to the job shop problem are discussed in [40] and [1]. Good results have been found for several of the classic test problems from [26] and [2]. However, none of the problems tested had all of the complexities found in the automaker data. Genetic algorithms have also demonstrated potential for solving scheduling problems.

Genetic algorithms (GAs) were introduced by [23] 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 [19], [13], or [27] for details.

GAs have been applied to scheduling problems by several authors. The primary strategy used is a literal permutation ordering encoding [3, 12, 28, 38]. Specialized operators have been developed to insure the feasibility of generated solutions including PMX Crossover [18], the subsequence-swap operator [21], other subsequence operators [12], edge recombination [41], order-based and position-based crossover [37], and forcing [28].

Two of the more innovative applications of GAs to scheduling problems are the Problem Space and Heuristic Space methods of [35]. Storer, Wu, and Vaccari [35] begin from the premise that most scheduling problems have base heuristics that are fast and obtain good solutions. The authors 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 [35] introduces another level of search. They combine several heuristics to create a superior meta-heuristic. These methods have been tested on several of the classical job shop test problems [36] and found to perform well.

Our approach is based on the random keys encoding of [8]. The random keys representation [8] 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 [33] where the authors embedded their encoding in a genetic algorithm that dynamically adjusts its internal representation of the search space according to the problem being solved.

The random keys GA (RKGA) operates in two spaces, the chromosome space and the schedule assignment space. For a five job single machine problem, the chromosome space consists of all points within a 5-dimensional hypercube, $\mathscr{C} = [0, 1]$.[5] Define chromosome $x \in \mathscr{C}$, where $x = \{x_1, x_2, \ldots, x_5\}$. Let $x_i$ represent the random key value for allele $i$ of chromosome $x$. The schedule assignment space, $\mathscr{S}$, contains all the possible schedules for the problem.

For a single machine problem with regular measure, this includes all the possible permutations of the jobs scheduled as early as possible. For the single machine problem, $\mathcal{C}$ is mapped to $\mathcal{S}$ by sequencing the jobs based on the sorted random key values. Consider a five-job problem. If a schedule were represented by the random key vector (.31, .12, .73, .91, .44), sorting to decode the schedule would result in the sequence 2, 1, 5, 3, 4. The initial random keys are randomly generated. During the progress of the algorithm, jobs that should be early in the sequence attain low keys and those which should be later in the sequence develop high keys.

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.

## 2. PROPOSED SOLUTION METHODOLOGY

We now extend the RKGA concept to include the following complexities: multiple, nonidentical machines, nonzero ready times, sequence dependent setups, tool constraints, and precedence. Changes are required in the encoding and in the function evaluation routine. We then describe the specific GA operators used in the algorithm. Finally, we demonstrate that there is a region with nonzero volume in the chromosome space $\mathcal{C}$ that maps to the set of optimal schedules. This leads to a simple proof of convergence.

### 2.1. Modifying the GA for Additional Problem Complexities

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 $i$.

After the random keys are sorted, the job sequence and machine assignments for each job have been determined. Nonzero ready times, sequence dependent setups, and tool availability can all be considered simultaneously in calculating a job's start 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 improves the overall objective. Where inserted idle time is not desirable, the RKGA will gradually select schedules without it.

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 in [29].

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.

### 2.2. Genetic Algorithm Operators

There are several types of genetic operators that may be used in the reproduction stage of a GA (see [19]). Because the random keys encoding preserves feasibility there is no need to use the specialized operators discussed in Section 1. 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 [19] 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 [34]) 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 $X = (x_1, x_2, \ldots, x_l)$ and $Y = (y_1, y_2, \ldots, y_l)$ be the $l$ random key alleles in these two chromosomes, respectively. Let $Z = (z_1, z_2, \ldots, z_l)$ be $l$ independent uniform (0, 1) variates and $P_c$ be the probability of crossover for each gene. Let $W = (w_1, w_2, \ldots, w_l)$ and $V = (v_1, v_2, \ldots, v_l)$ be the $l$ random key alleles for the two offspring that will result from the crossover of the two parent chromosomes. Determine $W$ and $V$ as

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

Since $P_c$ is not necessarily 0.5, this procedure can be used to bias selection toward one parent. We have found $P_c = 0.7$ to work for many problems. It appears to support development of a generalized form of building blocks.

Implementing a mutation operator for a real coded GA presents difficulties. As a result, we use immigration as described in [8]. It involves random generation of at least one chromosome each generation. Since our implementation of elitism only copies a few members from one generation to the next, immigrants are not automatically eliminated via that operator. Then the random parent selection mixes the new genetic material from the immigrants into the population in subsequent generations. In this way immigration maintains diversity as in traditional mutation.

A type of tournament selection [10], 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.

Empirical results indicate that if the GA does not improve the current best solution for 30 successive generations then additional searching yields little improvement. For this study, population convergence is assumed if the best solution does not change for 30 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 immigrants.
   - 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.

### 2.3. Mapping from $\mathscr{C}$ to $\mathscr{S}$

It is essential that the random keys encoding ensure that there exists a region in $\mathscr{C}$, with nonzero volume, that the evaluator maps to the optimal schedule set in $\mathscr{S}$. Then a search for this region in $\mathscr{C}$ can be used as a surrogate for a search of $\mathscr{S}$ for an optimal schedule. Constructing schedules in the manner described in Section 2.1, Theorem 1 shows that such a region exists within $\mathscr{C}$ for problems with a regular measure. We assume that there is no job preemption.

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

| | |
|---|---|
| $i$ | Index of job, $i \in \{1, 2, \ldots, n\}$. |
| $\alpha_i$ | The random key value for job $i$, $\alpha_i \in [0, 1]$ for all $i = 1, 2, \ldots, n$. |
| $[i]$ | Order statistic notation for the jobs from a sort of the $\alpha_i$ values. |
| $j, k$ | Indices of machines, $j, k \in M_i$. |
| $TO_i$ | 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.
$r_i$ The ready time for job $i$.
$d_i$ The due time for job $i$.
$p_i$ The processing time for job $i$.
$M_i$ The set of machines that can perform job $i$.
$d$ The total number of tools in the problem.
$SA$ The set of semiactive schedules.

REMARK: $TO_i$ is assumed scalar in the following proofs but is readily generalizable to a set.

The RKGA uses the procedure $\gamma(x)$, described below, to map from $\mathscr{C}$ to $\mathscr{S}$.
Let $x \in \mathscr{C}$ 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 $MA_k = 0$, $LT_k = \varnothing \quad \forall k = 1, 2, \ldots, m$.
Set $TA_h = 0 \quad \forall h = 1, 2, \ldots, d$.
For $(i = 1; i \leq n; i + +)$
{
if $TO_{[i]} = LT_{m_{[i]}}$ and $MTL_{TO_{[i]}} = m_{[i]}$
Setuptime $= 0$
else Setuptime $=$ Setupvalue
$S_{[i]} = max\{max\{MA_{m_{[i]}}, TA_{TO_{[i]}}\} + Setuptime, r_{[i]}\}$
$\mathrm{MA_{m_{[i]}}} = \mathrm{TA_{TO_{[i]}}} = \mathrm{S_{[i]}} + \mathrm{p_{[i]}}$
$LT_{m_{[i]}} = TO_{[i]}$
$\mathrm{MTL_{TO_{[i]}}} = \mathrm{m_{[i]}}$
}

The outcome of $\gamma$ is a set of start times, $S_i$, that combines with the machine assignments from the random keys to describe a schedule. Using the definition of $\gamma$, the following two results are readily verifiable (see [29]).

LEMMA 1: For all $x \in \mathscr{C}$, $\gamma(x)$ is a feasible schedule.

LEMMA 2: The image of $\mathscr{C}$ under $\gamma$ is exactly the set of semiactive schedules.

In Theorem 1 we prove that the GA finds an optimal solution with probability 1. Note that the algorithm presented in Section 2.2 generates a random sample at least once per generation. Let $x^t$ be the best solution found after $t$ generations and $Y \subset \mathscr{S}$ be the set of optimal schedules. The driver of the proof of Theorem 1 is the random generation of an immigrant each generation. The Theorem establishes that there is some point in $\mathscr{C}$ that maps to an optimal schedule in $\mathscr{S}$. We then show that there is an open ball ($n$-dimensional sphere of positive volume) in $\mathscr{C}$, centered at the given point, such that all points in the ball also map to that optimal solution. Since this ball has positive volume, the random sampling of immigration will eventually hit it, producing the optimal schedule.

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

PROOF: For a regular measure, [5] proved $(Y \cap SA) \neq \varnothing$. By Lemma 2, $\gamma^{-1}(Y \cap SA) \neq \varnothing$. Let $x \in \gamma^{-1}(Y \cap SA)$. Then $x = \{(m_i, \alpha_i)\}_{i=1}^{n}$, where $m_i \in M_i$ and $\alpha_i$ 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 $\alpha_i$ are generated from a uniform (0, 1), $\Delta > 0$ almost surely. Define the neighborhood of $x$, $N(x) = \{x' : m_i' = m_i, \quad |\alpha_i' - \alpha_i| < \Delta/2 \quad \forall i = 1, 2, \ldots, n\}$. If $x' \in N(x)$, then $\gamma(x') = \gamma(x) \in Y$. When randomly generating $x' \in \mathcal{C}$,

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

Now, in each generation, the immigrant, $x$, which is uniformly generated from $C$, 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 $\gamma$ 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. COMPUTATIONAL RESULTS

To determine the effectiveness of the RKGA for scheduling complex settings, we tested two data sets of actual problems from an automaker. We are not aware of additional data sets in the literature that contain the types of complications found in the automaker data. To demonstrate robustness, tests on randomly generated makespan job shop problems are referenced.

### 3.1. Real Auto Problems

In this section we will describe the problems against which the algorithm was tested, describe the testing process, and present results.

The first and second data sets comprise a total of twelve problems from a large automaker [9]. The seven problems in the first data set each contain approximately 360 jobs that are to be scheduled on two identical machines. Recall from Section 1 that these problems contain the following complexities: nonzero ready times, due times, 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 is 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 with approximately 270 jobs to be scheduled on 10 nonidentical 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 one.

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 can be compared to total tardiness lower bounds for each of the problems. A lower bound for each problem is

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

The RKGA's results are compared to the three scheduling methods presented in [9]: dynamic dispatching rules (DISP), a multiple choice integer programming based method (MCIP), and matchup scheduling (MUSA). In addition, we provide comparisons with two additional dispatching heuristics that we developed and with random search. These additional heuristics utilize the Earliest Due Date (EDD [24]) and Modified Due Date (MDD [6]) list processing rules. These list processing rules provide a job sequence but do not provide job to machine assignments. Therefore, a GA was used to determine good job to machine assignments given a job sequence that resulted from a list processing rule. The mechanics and parameter settings of the GA were similar to the RKGA except that the job sequencing was based on the dispatching rule and the GA only determined the machine assignments.

The results produced by any run of a GA are a function of the random seed used. It is disconcerting if the same algorithm, on the same data set, can perform well or poorly depending on the random seed used. To study this issue, the GA was run 10 times for each problem using a different random seed for each run. The minimum, median, and maximum values over the 10 random seeds are presented. The minimum and maximum values provide best and worst case performance measures. The median is used rather than the mean in order to reduce the effect of extreme minimum and maximum values.

In all runs the following parameter settings were used: *population_size* = 1024, *number_of_clones* = 60, *number_of_immigrants* = 40, *maximum_number_of_generations* = 500.

In order to assure fair comparisons, each competitor must have at least the same computation resources available to it as was provided the RKGA. The DISP, MCIP, and MUSA algorithms were run to their logical conclusions (e.g., the branch-and-bound ran to optimal stop). Hence, additional computation time would not have changed the results of these methods. The EDD- and MDD-based heuristic methods and the random search were all permitted to evaluate the maximum number of solutions that could be investigated by the RKGA. However, the RKGA always converged prior to examining this maximum quantity. Thus these other methods were permitted to examine more solutions and utilize more computation time.

From Table 1 we can compare the lower bound for each of the 12 problems in the first and second data sets with the best solution obtained using the GA. The results indicate that solutions within 0.4–6.6% of the lower bound can be found for the seven disruptions of data set one. For the five problems in the second data set the percent deviation from the lower bound is not meaningful because the lower bound is zero for four of the five instances. However, the maximum total tardiness was less than 23 units above the lower bound for all seeds tested across

**Table 1.** Automaker data test results.[a]

| Problem | Tardiness lower bound | RKGA Min | RKGA Med | RKGA Max | DISP Best of four dispatching rules | MCIP | MUSA | EDD GA mach. assignment (min) | MDD GA mach. assignment (min) |
|---|---|---|---|---|---|---|---|---|---|
| 1.1 | 351.0 | 369.7 | 371.2 | 374.2 | 438.0 | 431.0 | 440.0 | 455.1 | 449.1 |
| 1.2 | 355.0 | 357.3 | 360.1 | 362.7 | 376.0 | 385.0 | 375.0 | 441.6 | 427.7 |
| 1.3 | 349.0 | 352.6 | 353.9 | 357.8 | 491.0 | 403.0 | 378.0 | 433.8 | 413.3 |
| 1.4 | 353.0 | 354.9 | 355.8 | 359.8 | 373.0 | 378.0 | 375.0 | 428.0 | 424.4 |
| 1.5 | 351.0 | 352.4 | 355.1 | 357.3 | 375.0 | 371.0 | 381.0 | 415.6 | 445.0 |
| 1.6 | 293.0 | 294.6 | 295.7 | 298.5 | 404.0 | 314.0 | 308.0 | 415.6 | 443.7 |
| 1.7 | 352.0 | 356.4 | 357.4 | 360.9 | 435.0 | 391.0 | 374.0 | 445.1 | 432.8 |
| Average | 343.4 | 348.3 | 349.9 | 353.0 | 413.1 | 381.9 | 375.9 | 433.5 | 433.7 |
| 2.1 | 0.0 | 0.0 | 0.4 | 5.8 | 0.0 | 0.0 | 0.0 | 19.7 | 22.7 |
| 2.2 | 0.0 | 0.0 | 2.6 | 5.8 | 0.0 | 0.0 | 0.0 | 11.9 | 26.3 |
| 2.3 | 0.0 | 0.0 | 0.0 | 14.5 | 4.0 | 112.0 | 31.0 | 22.0 | 12.1 |
| 2.4 | 12.9 | 13.3 | 15.8 | 35.1 | 184.0 | 600.0 | 139.0 | 56.2 | 49.1 |
| 2.5 | 0.0 | 0.0 | 5.1 | 9.5 | 9.0 | 9.0 | 9.0 | 20.6 | 7.7 |
| Average | 59.4 | 60.3 | 62.3 | 70.6 | 101.7 | 183.8 | 92.5 | 94.0 | 91.9 |

[a] Actual total tardiness values are shown for each method.

all the problems. The data in Table 1 also indicate that the RKGA was consistent across the different random number seeds.

Table 1 provides a comparison between the RKGA and the solution methods found in [9]. For data set one, the RKGA consistently found solutions that were better than those found using alternative methods. The median (across the seeds) solution found by the RKGA was on average 18% better than that found using DISP, 9% better than that found by MCIP, and 7% better than those found by MUSA. In addition to exhibiting the best average performance across the problems in data set one, the data of Table 1 also indicate that the RKGA was the best performer for every problem instance.

For data set two the RKGA also demonstrated better average performance than DISP, MCIP, and MUSA. On average, the solutions found by the DISP, MCIP, and MUSA methods have, respectively, 63%, 195%, and 48%, more total tardiness than the median RKGA solution. Note that these percentages are much larger than those for data set one because the actual total tardiness values for the problems of data set two are much smaller numbers. However, the RKGA is clearly better on average.

Analyzing all twelve problems of the two data sets together, statistical testing using a paired *t*-test found that the differences for DISP and MUSA were significant at alpha levels of .012 and .019, respectively. If all 12 problems are considered, the alpha level for MCIP is .130. This may seem counterintuitive since MCIP provides larger total tardiness values, in general, than MUSA. The cause of the large alpha level value is the large tardiness on problem four of the second data set. This large tardiness value results in a large standard deviation in the difference between the RKGA median and MCIP. This results in a smaller value of the test statistic which leads to the large alpha value. If this problem is dropped and the paired *t*-test is rerun with only the remaining 11 problems, then the difference between the RKGA median and MCIP is significant at an alpha level of .012. Thus, eliminating problem four of the second data set, which provides the best example of the RKGA outperforming the MCIP, actually reduces the alpha level. This is a good example of how statistical significance can differ from practical significance.

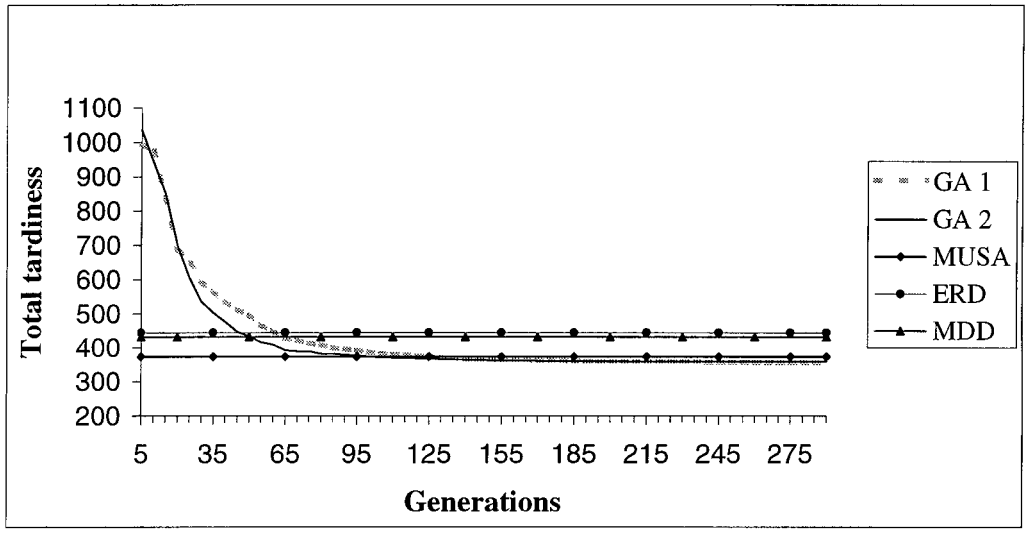


**Figure 1.** Convergence graph for the RKGA.

Table 1 shows only the best result found over all 10 seeds for both the EDD and MDD rules. For data set one both methods find solutions that on average have 24% higher total tardiness than the average median seed value of the RKGA. For the second data set the average deviation increases to 51% and 48%, respectively, for the EDD- and MDD-based heuristics. A paired *t*-test indicated that there was a difference between the median RKGA solution, and the EDD- and MDD-based heuristics that was significant at alpha levels of .0002 and .0005, respectively. Note that these methods exhibited much more variation over the random seeds than the RKGA.

We also compared the results of the RKGA with a random search. Due to the complexity of the problem random search performed very poorly. Results are not listed in Table 1 but can be summarized. For data set one, the random search found solutions with total tardiness that was 2.5–5 times more than the RKGA. For data set two, the solutions found had total tardiness on the order of 10,000–15,000 as opposed to 35 or less for the RKGA.

The RKGA was able to solve all 12 problems of data sets one and two, for a single random number seed, on average in 9 min on a Sun Ultra 2, 200-MHz processor. Figure 1 compares the convergence graph of the RKGA with the best solutions found by the MUSA, EDD, and ERD algorithms. GA 1 and GA 2 represent the convergence graphs of two individual seeds for problem 1.7. The graph shows that the RKGA has a relatively smooth convergence graph and quickly finds solutions that are within 5–10% of optimal.

Test results for data set one show that the RKGA 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 RKGA 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 second data set because the lower bound is 0 for four of the five problems.

### 3.2. Makespan Job Shops

The robust nature of the RKGA has been verified by conducting additional testing on the $n/m/J/C_{max}$ problem. In this problem there are $n$ jobs to be processed on $m$ machines. Each job

contains $m$ operations and must visit each of the $m$ machines. The operations must be completed in a specific order, and each operation can only be completed on one machine. Therefore, there is a strict precedence ordering of the machine sequence for each job. However, the machine sequences for different jobs may vary. The objective is to minimize the completion time of the job that completes last.

The $n/m/J/C_{max}$ problem represents a significant departure from the automaker problems since there are no tooling constraints, alternative routings, sequence dependent setup times, release times, or due times. However, the $n/m/J/C_{max}$ problem is still NP-hard (see, for example, [32]) and represents a difficult scheduling problem that has been analyzed by a number of researchers [29]. Experiments were conducted using the well-known data sets from [26]. These problems contain 10–30 jobs and 5–15 machines. The RKGA found solutions that were on average within 1–2% of the best known solutions. Details of the implementation and comparison can be found in [30]. The good performance of the RKGA on the $n/m/J/C_{max}$ problem provides further validation of this solution approach and demonstrates that the methodology can be applied to multiple problem contexts.

## 4. SUMMARY AND CONCLUSIONS

We have presented a scheduling algorithm that utilizes the random keys encoding within the context of a GA. This representation and algorithm are proven to find an optimal solution if run long enough. 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 consistent results across the different random seed values. The median (over the 10 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 5 units of the tardiness lower bound for the second data set. The maximum deviation from the lower bound over all seeds was less than 6.6% for all seven problems of data set one and 23 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 other scheduling procedures that have been applied to these problems, given equivalent computational resources. Thus, the RKGA provides a good method for generating schedules for problems with the complexities found in the automaker data sets.

The robust nature of the RKGA was discussed also. The same methodology used for the automaker problems yielded high quality solutions when applied to the $n/m/J/C_{max}$ problem. Additionally, the RKGA can also be extended to include other problem complexities not found in the automaker data such as limited buffer space for each machine. The RKGA methodology can also be applied to other objective function measures. Combined with the uniformity of performance over random seeds and reasonable computation times, these results suggest that the RKGA could be useful in an application setting.

## 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

[1] E.H.L. Aarts, P.J.M. Van Laarhoven, J.K. Lenstra, and N.L.J. Ulder, A computational study of local search algorithms for job shop scheduling, ORSA J Comput 6 (1994), 118–125.
[2] J. Adams, E. Balas, and D. Zawack, The shifting bottleneck procedure for job shop scheduling, Manage Sci 34 (1988), 391–401.
[3] S. Bagchi, S. Uckun, Y. Miyabe, and K. Kawamura, Exploring problem-specific recombination operators for job shop scheduling, Proc Fourth Int Conf Genetic Algorithms, 1991, pp. 10–17.
[4] K. Baker, Introduction to sequencing and scheduling, Wiley, New York, 1974.
[5] K. Baker, Elements of sequencing and scheduling, Amos Tuck School of Business Administration, Dartmouth College, Hanover, NH, 1995.
[6] K. Baker and J. Kanet, Job shop scheduling with modified due dates, J Operations Manage 4 (1983), 11–22.
[7] J.W. Barnes and M. Laguna, A tabu search experience in production scheduling, Ann Operations Res 41 (1993), 141–156.
[8] J.C. Bean, Genetics and random keys for sequencing and optimization, ORSA J Comput 6 (1994), 154–160.
[9] J.C. Bean, J.R. Birge, J. Mittenthal, and C. Noon, Matchup scheduling with multiple resources, release dates and disruptions, Operations Res 39 (1991), 470–483.
[10] A. Brindle, Genetic algorithms for function optimization, Doctoral dissertation and Technical Report TR 81-2, University of Alberta, Department of Computer Science, Edmonton, 1981.
[11] V. Černy, Thermodynamical approach to the traveling salesman problem: an efficient simulation algorithm, J Optim Theory Appl 45 (1985), 41–51.
[12] G.A. Cleveland and S.F. Smith, Using genetic algorithms to schedule flow shop releases, Proc Third Int Conf Genetic Algorithms, 1989, pp. 160–169.
[13] L. Davis Handbook of genetic algorithms, Van Nostrand, New York, 1991.
[14] M. Dell'Amico and M. Trubian, Applying tabu search to the job-shop scheduling problem, Ann Operations Res 41 (1993), 231–252.
[15] J. Du and J.Y.-T. Leung, Minimizing total tardiness on one machine is NP-hard, Math Operations Res 15 (1991), 483–495.
[16] F. Glover, Tabu search—part I, ORSA J Comput 1 (1989), 190–206.
[17] F. Glover, Tabu search—part II, ORSA J Comput 2 (1990), 4–32.
[18] D.E. Goldberg and R. Lingle Jr., Alleles, loci, and the traveling salesman problem, Proc First Int Conf Genetic Algorithms, 1985, pp. 154–159.
[19] D.E. Goldberg, Genetic algorithms in search optimization and machine learning, Addison Wesley, Reading, MA, 1989.
[20] J. Grabowski, E. Nowicki, and S. Zdrzalka, A block approach for single-machine scheduling with release dates and due dates, Eur J Operational Res 26 (1986), 278–285.
[21] J.J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, Genetic algorithms for the traveling salesman problem, Proc First Int Conf Genetic Algorithms, 1985, pp. 160–168.
[22] D.J. Hoitomt, P.B. Luh, and K.R. Pattipati, A practical approach to job-shop scheduling problems, IEEE Trans Robot Automat 9 (1993), 1–13.
[23] J.H. Holland, Adaptation in natural and artificial systems, The University of Michigan Press, Ann Arbor, 1975.
[24] J.R. Jackson, Scheduling a production line to minimize maximum tardiness, Research Report 43, Management Sciences Research Program, UCLA, Los Angeles, 1955.
[25] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, Optimization by simulated annealing, Science 220 (1983), 671–680.
[26] S. Lawrence, Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement), Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, 1984.
[27] Z. Michalewicz, Genetic algorithms + data structures = evolution programs, Springer-Verlag, New York, 1994.
[28] R. Nakano, Conventional genetic algorithm for job shop problems, Proc Fourth Int Conf Genetic Algorithms, 1991, pp. 474–479.
[29] B.A. Norman, The random keys genetic algorithm for complex scheduling problems, Ph.D. dissertation, University of Michigan, Ann Arbor, 1995 (unpublished).

[30] B.A. Norman and J.C. Bean, A random keys genetic algorithm for job shop scheduling problems, Eng Des Automat 3 (1997), 145–156.
[31] E. Nowicki and C. Smutnicki, A fast taboo search algorithm for the job shop problem, Manage Sci 42 (1996), 797–813.
[32] M. Pinedo, Scheduling theory, algorithms, and systems, Prentice Hall, Englewood Cliffs, NJ, 1995.
[33] C.G. Shaefer and S.J. Smith, The ARGOT strategy II: combinatorial optimizations, Technical Report, Thinking Machines Corporation, Burlington, Massachusetts, 1988.
[34] W.M. Spears and K.A. De Jong, On the virtues of parameterized uniform crossover, Proc Fourth Int Conf Genetic Algorithms, 1991, pp. 230–236.
[35] R.H. Storer, S.D. Wu, and R. Vaccari, New search spaces for sequencing problems with application to job shop scheduling, Manage Sci 38 (1992), 1495–1509.
[36] R.H. Storer, S.D. Wu, and I. Park, Genetic algorithms in problem space for sequencing problems, Proc Joint US-German Conf Operations Res Prod Plan Control, 1992, pp. 584–597.
[37] G. Syswerda, "Schedule optimization using genetic algorithms," Handbook of genetic algorithms, L. Davis (Editor), Van Nostrand, New York, 1989, pp. 332–349.
[38] G. Syswerda, The application of genetic algorithms to resource scheduling, Proc Fourth Int Conf Genetic Algorithms, 1991, pp. 502–508.
[39] E. Taillard, Parallel tabu search techniques, ORSA J Comput 6 (1994), 108–117.
[40] P.J.M. Van Laarhoven, E.H.L. Aarts, and J.K. Lenstra, Job shop scheduling by simulated annealing, Operations Res 40 (1992), 113–125.
[41] D. Whitley, T. Starkweather, and D. Fuquay, Scheduling problems and traveling salesman: the genetic edge recombination operator, Proc Third Int Conf Genetic Algorithms, (1989), pp. 133–140.
[42] M. Widmer, Job shop scheduling with tooling constraints: a tabu search approach, J Operational Res Soc 42 (1991), pp. 75–82.