Minimizing Total Flowtime and Maximum Earliness on a Single Machine Using Multiple Measures of Fitness

Mary E. Kurz
Department of Industrial Engineering
Clemson University
110 Freeman Hall
Clemson, SC 29634-0920
864-656-4652
mkurz@clemson.edu

Sarah Canterbury
Department of Industrial Engineering
Clemson University
110 Freeman Hall
Clemson, SC 29634-0920
scanter@clemson.edu

ABSTRACT
The intent of this research is to investigate methods to use genetic algorithms to find the set of efficient solutions to a bi-criteria problem. We propose a general methodology which is characterized by using different criteria upon which the decision to retain chromosomes into the next generation is made. We perform elite reproduction based on two general measures of "eliteness": non-dominated in the current population and performance measured in terms of each criterion individually. We investigate its performance on a specific bi-criteria scheduling problem, minimizing total flowtime and maximum earliness on a single machine.

Categories and Subject Descriptors
I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search – *heuristic methods*, *scheduling*

General Terms
Algorithms, Experimentation

Keywords
Multicriteria genetic algorithm, bi-criteria scheduling

1. INTRODUCTION
The intent of this research is to investigate methods to use genetic algorithms to find the set of efficient solutions to a bi-criteria problem. We propose a general methodology which is characterized by using different criteria upon which to keep chromosomes. We perform elite reproduction based on two general measures of "eliteness": non-dominated in the current population and performance measured in terms of each criterion individually. We investigate its performance on a specific bi-criteria scheduling problem, minimizing total flowtime and maximum earliness on a single machine. We begin with a brief review of genetic algorithms and describe the random keys genetic algorithm (RKGA) in Section 2. In Section 3, we introduce a bi-criteria RKGA, BRKGA. In Section 4, we describe a bi-criteria single machine scheduling problem and the problem specific decisions for BRKGA, as well as a heuristic for the problem from the literature. Computational results are described in Section 5, with conclusions and areas for future research described in Section 6.

2. GENETIC ALGORITHMS
GAs were introduced in 1975 by Holland [5], who provided the basic framework: chromosomes represent solutions that reproduce based on how well they solve the problem at hand in a manner analogous to survival of the fittest. The chromosomal representation of a solution is an important design feature of a genetic algorithm. Often, the chromosome is a string of 0s and 1s; however, other possibilities exist, such as strings of non-negative integers. The chromosomal representation of the solution information can take many forms and influences the types of genetic operators. Particularly readable introductions to GAs are the texts by Michalewicz [11] and Goldberg [3]. Research reported in the literature indicate the power and the limitations of GAs, especially in the context of scheduling (for example, [1], [9], [10], [15]). While we have seen the greatest use of GAs in single criterion optimization, more researchers are using GAs in multi-objective scheduling (for example, [6], [12], [13]).

Bean [1] has introduced an alternative method to encode problem solutions using random numbers called a Random Keys Genetic Algorithm (RKGA), which has been applied to numerous problems. Recent applications include flexible flowlines with sequence-dependent setup times [10] and batch-processing machine scheduling problems [15]. RKGA differs from traditional genetic algorithms most notably in the solution representation. Random numbers serve as sort keys used to decode the chromosome into a solution; all chromosomes can be decoded into feasible solutions, resolving a problem common with other representations (such as permutation) sometimes used for scheduling. The decoded solution is evaluated with a fitness function that is appropriate for the problem at hand. For example, Norman and Bean [12] suggest using the following solution representation for an identical multiple machine scheduling problem. Each job is assigned a real number whose integer part is the machine number to which the job is assigned and whose fractional part is used to sort the jobs assigned to each machine. Once the job assignments and order on each machine is found through the decoding, a schedule can be built incorporating additional factors such as nonzero ready times and sequence-dependent setup times. An example chromosome for 6 jobs with 2 machines is shown in Figure 1. Decoding this chromosome indicates that jobs 2 and 4 will be processed by the first machine and jobs 1, 3, 5, and 6 will be processed by the second machine. Jobs 2 and 4 are assigned to the same machine in that order since 0.12 is less than 0.57. The order for machine 2 is found in a similar manner. The sequence is shown in Figure 2.

Job	**1**	**2**	**3**	**4**	**5**	**6**
Gene	2.86	1.12	2.34	1.57	2.78	2.23

Figure 1: RKGA Chromosome Example

Machine 1	2	4		
Machine 2	6	3	5	1

Figure 2: RKGA Chromosome Sequence Example

The genetic operators and related parameters used in Bean [1] are summarized in the following. The initial population is generated randomly. An elitist strategy is used for reproduction. Each chromosome is decoded and the resulting solution is evaluated for the performance measure. The 20% "best" chromosomes are automatically copied to the next generation. Parametrized uniform crossover is used to select the next 79% chromosomes in the next generation. For each chromosome in the next generation, the following is performed. Two chromosomes in the current generation are selected at random. For each gene, a random number is generated. If the value is less than 0.7, the value from the "first" chromosome is copied to the new chromosome, otherwise the value from the "second" chromosome is selected. Figure 3 illustrates the crossover method.

Chromosome 1	<u>23</u>	<u>03</u>	<u>45</u>	89
Chromosome 2	15	45	85	<u>03</u>
Crossover Random Number	0.45	0.23	0.68	0.75
New Chromosome	23	03	45	03

Figure 3: Example of parametric uniform crossover

The remaining 1% of the next generation is filled through "immigration", in which new chromosomes are randomly generated. The above procedures are repeated until 100 iterations have been performed without finding an improved schedule. The values reported here were selected empirically for the particular problem at hand.

The concept of *efficient solutions* is used to identify solutions that may be desirable to a decision maker. A solution σ is weakly efficient (with respect to the criteria of interest, here called z_1 and z_2, which are to be minimized) if there is no other solution σ' such that $z_1(\sigma) < z_1(\sigma')$ and $z_2(\sigma) < z_2(\sigma')$. A solution σ is efficient if it is weakly efficient and at least one of the previous relations holds as a strict inequality. We then say that solution σ' dominates solution σ. These definitions are easily extendable to the multi-criteria case. The use of GAs in finding the set of efficient solutions seems especially desirable because GAs evolve sets of solutions and multi-criteria problem solvers may want the entire set of efficient solutions. Since it is not guaranteed that any heuristic will find optimal solutions to NP-hard single criterion optimization problems, GAs can only find *approximately efficient solutions* (AES) to multi-criteria optimization problems (this terminology is used in [7]).

T'Kindt and Billaut [14] provide a survey of multicriteria scheduling problems, organized by the complexity of the problem, if known. Since the survey is organized in this way, it is hard to see easily for which problems genetic algorithms or other specific techniques have been attempted. Coello Coello and Mariano Romero [2] provide a survey of evolutionary algorithms used in multicriteria optimization, with analysis and identification of future directions. Through these surveys and the literature citing RKGA, we see that Bean's RKGA has not been previously applied to bi-criteria scheduling problems. We select the RKGA type of encoding due to the fact that it always decodes into a feasible sequence.

3. BICRITERIA RKGA (BRKGA)

This work primarily is concerned with the adaptation of RKGA to a bi-criteria application. The general procedure is outlined below and strongly resembles the basic RKGA methodology. In the interest of space, we focus our attention on those steps that differ from RKGA, namely steps 1, 4, 5, 6 and 7, and discuss these elements in detail below.

1. Determine an upper bound for the number of efficient solutions to the problem, if possible.
2. Initialize the population.
3. Evaluate the chromosomes in the population. Store the values of each criterion separately.
4. Determine which chromosomes represent potential efficient solutions. Call them the AES chromosomes.
5. Perform multi-criteria elite reproduction.
6. Perform parametric uniform crossover.
7. Perform immigration.
8. Evaluate the chromosomes in the population.
9. Find the AES chromosomes.
10. If the stopping conditions have been met, STOP. Otherwise, go to step 5.

3.1 Maximum Number of Efficient Solutions

As our overall intent is to identify the set of efficient solutions, we should ensure that the population is large enough to hold the entire set. If the objectives can be optimized individually and the data is integer-valued, we can deduce that the set of efficient solutions will have no more elements than one per integer point between and including the extreme values of the individual objective functions.

3.2 Identifying Approximately Efficient Solutions

Because we are using a genetic algorithm, we cannot be sure that a solution is truly an efficient solution. However, we can determine if some solution in the current population is dominated by other solutions. If a solution in the current population is not strictly dominated by any other solution in the current population, we will mark it as an approximately efficient solution (AES).

3.3 Multi-Criteria Elite Reproduction

We propose using two types of reproduction operations to perform multi-criteria elite reproduction. This is the primary contribution of this methodology. The first retains all of the AES chromosomes for the next population. In this way, BRKGA can always keep the most efficient chromosomes seen so far. The second type will be traditional elite reproduction with respect to each of the criteria, considering only the chromosomes that have not already been selected for reproduction because of their AES status. We refer to the maximum number of chromosomes kept from each of these traditional elite reproductions as Zi_MAX_KEPT, replacing "Zi" by the criteria of interest later. This reproduction operator is inspired by the idea of *population isolation* [4], where two groups of organisms are unable to mate because they are separated by some physical distance or barrier. In BRKGA, the chromosomes were chosen to survive to the next generation purely based on a one dimensional measure of their worth so in some ways, we performed *criterion* isolation.

3.4 Parametric Uniform Crossover

The parametric uniform crossover operator described above will be used. However, the maximum number of chromosomes for the new generation created using crossover will be set at MAX_CROSS. If many solutions were kept in the AES reproduction step, less than MAX_CROSS new chromosomes may be created, in order that the new population has at most POP_SIZE members.

3.5 Immigration

If less than POP_SIZE members are in the new population following the parametric uniform crossover step, the remaining members of the new generation will be formed through immigration. Immigration may not be performed in every iteration of BRKGA.

4. A BI-CRITERIA SCHEDULING PROBLEM

This work addresses scheduling *n* jobs on a single machine to minimize total flowtime and maximum earliness. The following data are required as input:

p_i = processing time of job *i*, *i*=1, …, *n*

d_i = due date of job *i*, *i*=1, …, *n*

The jobs are assumed to all be available at time 0 and all data is integer. The work considered here does not allow idle time to be inserted between jobs, so a single permutation of the *n* jobs is sufficient to completely compute a schedule. Let σ denote a sequence of the *n* jobs. The completion time of job *i* in sequence σ, $C_i(\sigma)$, is the sum of its processing time as well as the processing times of all the jobs scheduled before it. The following values can be computed:

$F(\sigma)$ = total flowtime of all jobs in sequence σ ;

$$F(\sigma) = \sum_{i=1,\dots,n} C_i(\sigma)$$

$E_i(\sigma)$ = earliness of job *i*, *i*=1, …, *n* in sequence σ ;

$$E_i(\sigma) = \max\left(0, d_i - C_i(\sigma)\right)$$

$E_{\max}(\sigma)$ = maximum earliness of all jobs in sequence σ ; $E_{\max}(\sigma) = \max_{i=1,\dots,n} \left\{E_i(\sigma)\right\}$

Minimizing the total flowtime subject to an upper bound on the maximum earliness (taking the epsilon-constraint approach) is strongly NP-hard [7]. As genetic algorithms have proved to be an effective method of finding good solutions to single criterion optimization problems, we propose the use of genetic algorithms in finding the set of efficient sequences. It is not guaranteed, however, that genetic algorithms will find the optimal solutions to single criterion optimization problems. For this reason, the proposed method can only find *approximately efficient sequences*.

In this section, several previously known results, to which reference will be made, are presented. Several papers have been published addressing this particular single machine bicriteria scheduling problem. This brief review only considers that work which most directly relates to this paper. First, it is well known that ordering jobs in non-decreasing order of processing times (SPT) is optimal for minimizing total flowtime on a single machine. [6], [7] and [8] consider minimizing both total flowtime and maximum earliness on a single machine. Koksalan *et al* [7] develop a heuristic to generate all approximately efficient sequences, based on the epsilon-constraint version of the problem (finding good flowtime solutions for a given maximum earliness value). They also

- Report that ordering jobs in non-decreasing order of slack times (MST) is optimal for minimizing the maximum earliness on a single machine with no inserted idle time. The slack time of job *i* is $d_i - p_i$.
- Prove that if $p_i < p_k$ and $d_i - p_i \le d_k - p_k$, then job *i* precedes job *k* in every efficient sequence.
- Give an upper bound on the number of efficient sequences.

Koksolan [8] locates some pre-defined number of non-dominated sequences using the heuristic in [7] as a sub-procedure. Karasakal and Koksalan [6] use simulated annealing to find the "best" total flowtime for a given maximum earliness value.

We note that the approaches of [6], [7] and [8] differ from that presented here in at least two ways. First, [6], [7] both solve epsilon constraint versions of the problem. Two extreme values for the maximum earliness are used to generate a list of all possible values for maximum earliness in the set of efficient solutions. Then, the heuristic or simulating annealing approach is used to find the "best" value of the total flowtime possible with each potential maximum earliness value. This is possible because there are known optimal sequences for each of the criteria considered individually. This approach may not extend to the case where neither criteria can be easily optimized when considered alone. [8] can only identify at most the preselected number of efficient sequences. We evaluate BRKGA in comparison to the heuristic approach proposed in [7].

4.1 Problem Specific Implementation Decisions

In this section we describe the problem specific decisions, such as how solutions are represented, how the maximum number of efficient solutions possible can be known for this problem, and stopping criteria.

4.1.1 Solution Representation, Decoding and Evaluation

As we are considering a single machine environment, it is sufficient to only use the random keys portion for sorting. We will use one integer for each job, which will be sorted in increasing order. Figure 4 illustrates a chromosome for a five job example and the resulting job order.

Chromosome	23	3	45	33	4
Job order	2	5	1	4	3

Figure 4: Example chromosome and decoding

Once the chromosome has been decoded, a job order results. This job order is then used to calculate the total flowtime and maximum earliness that results from the schedule. Recall that Koksalan *et al* [7] show that if $p_i < p_k$ and $d_i - p_i \le d_k - p_k$, then job *i* precedes job *k* in every efficient sequence. This rule is applied to every job order as it is evaluated; the resulting interchanges are also applied to the chromosome that generated the job order. In this way, we utilize the problem specific information.

4.1.2 Maximum Number of Efficient Solutions

In this test application, either criterion can be optimized individually with simple rules; SPT (shortest processing time) for the total flowtime and MST (minimum slack time) for the maximum earliness, as described earlier. Koksalan *et al* [7] state that the maximum number of efficient solutions is bounded by the minimum of two values. Let *Emax*(*MST*) be the (optimal) maximum earliness found by utilizing the MST sequence and *Emax*(*SPT*) be the maximum earliness found by utilizing the SPT sequence *and* breaking ties in processing times with the MST rule. Also, let *F*(*SPT*) be the total flowtime found utilizing the SPT sequence. They note that an efficient sequence with the maximum total flowtime value *F*(*P*) can be found by solving the problem **P**: minimize the total flowtime subject to the constraint that the maximum earliness is exactly *Emax*(*MST*). Despite the fact that this problem cannot be solved easily because it is NP-hard, we do know that the number of efficient sequences is no more than $\min\{F(P) - F(SPT) + 1, Emax(SPT) - Emax(MST) + 1\}$.

Because the range of flowtime values is much larger than the range of maximum earliness values, we will use MAX_ES= $Emax(SPT) - Emax(MST) + 1$ as the maximum number of efficient sequences.

4.1.3 Population Initialization

The initial population is generated randomly. The population size will be constant after every generation has been created and will be called POP_SIZE. In the case that MAX_ES>POP_SIZE (meaning that the population may not be able to hold one AES per potential efficient solution), we will use POP_SIZE = 2*MAX_ES.

4.1.4 Reproduction

We propose using three reproduction operations. The first retains all of the AES chromosomes for the next population. In this way, BRKGA can always keep the most efficient chromosomes seen so far. The other reproduction operations will be traditional elite reproduction with respect to each of the criteria, considering only the chromosomes that have not already been selected for reproduction because of their AES status. We will set the maximum number of chromosomes kept from each of these traditional elite reproductions at FT_MAX_KEPT and EMAX_MAX_KEPT.

4.1.5 Stopping Criteria

The stopping criteria were selected to be evaluating a maximum number of chromosomes MAX_EVAL or the number of AES's equals MAX_ES. The maximum number of chromosome evaluations is a common stopping criteria; the variation seen in different GA runs which use this stopping criteria can be used to evaluate whether the GA appears to be run long enough or not. The second criteria was decided upon in the faint hope that many, many AES's would be generated. While the first type of reproduction keeps all AES's in the current population, the number of AES's may not increase monotonically through the run of BRKGA. Since MAX_ES is an upper bound on the number of true efficient solutions, it doesn't seem unreasonable that finding that many AESs would indicate that BRKGA had done a good job.

4.2 Heuristic by Koksalan, Azizoglu and Kondacki

In [7] (pg. 194), a heuristic is given to "generate all approximately efficient sequences" for the problem to minimize the flowtime and maximum earliness on a single machine. We call this procedure "KAK" and repeat the steps here for the convenience of the reader.

```
1. Calculate Emax(MST), Emax(SPT) and let k=1. If
   Emax(MST) = Emax(SPT), stop. Otherwise, let E =
   Emax(SPT) – 1.
2. Find an AES to the problem P: Min F subject to Emax ≤ E
   using the following rule:
      For t = 1 to n, Let $S_t$ be the set of jobs whose earliness
      values do not exceed E when assigned to position t given
      the set of jobs assigned to the first t-1 positions. Let
      $i^* = \arg\min\{p_i : i \in S_t\}$, breaking ties according to the
      MST rule. Assign i* to position t.
   Let k = k +1 and Emax* be the maximum earliness of the
   sequence found. If Emax* = Emax(MST), stop. "All distinct
   AESs are generated"; there are k of these. Otherwise, go to
   Step 3.
3. Let E = Emax* – 1 and go to Step 2.
```

5. COMPUTATIONAL EXPERIENCE

In order to investigate the proposed BRKGA, a small set of experiments was undertaken. The data generated has the same characteristics as that in [7]. Processing times have either low or high variability: times are taken from a discrete uniform distribution [1,10] for the low processing time files and from a discrete uniform distribution [1,30] for the high processing time files. Due dates are generated after the processing times are generated from a discrete uniform distribution $\left[\left(1-\tau-R/2\right)\sum p_i, \left(1-\tau+R/2\right)\sum p_i\right]$ where τ (controlling tardiness) and R (controlling the due date range) take on different values, as shown in Table 1. 10 problem instances with 50, 100, 150 and 200 jobs each are generated, resulting in 400 problem instances total.

Table 1: Problem set definitions

Problem set	τ	R
I	0.20	0.60
II	0.20	1.00
III	0.50	0.60
IV	0.50	1.00
V	0.65	0.20

All algorithms were implemented in C, compiled with Microsoft Visual C++ and run on a PC with a Pentium IV 2.8 GHz processor with 512 MB of RAM. KAK was run once per problem, as it is a deterministic algorithm. BRKGA was run on each problem five times with different random number seeds, with the parameter values shown in Table 2. We also ran BRKGA once with a much higher number of evaluations allowed – ten times MAX_EVAL chromosome evaluations were allowed. The total running time for one run of the 400 files was over 52 hours, but we provide the results of this single run for comparison purposes. We refer to the BRKGA results with the lower number of maximum evaluations as "Short BRKGA" and the BRKGA results with the higher number of maximum evaluations as "Long BRKGA".

Table 2: BRKGA settings

POP_SIZE	Min(400, 2*MAX_ES)
FT_MAX_KEPT	0.10 * POP_SIZE
EMAX_MAX_KEPT	0.10 * POP_SIZE
MAX_CROSS	0.70 * POP_SIZE
MAX_EVAL (short)	500 * POP_SIZE
MAX_EVAL (long)	10 * 500 * POP_SIZE

It is difficult to assess the performance of heuristics on multi-criteria problems, as discussed by Coello Coello and Mariano Romero [2]. In order to assess the effectiveness of BRKGA, we compiled the following information. We report all performance as a percentage of MAX_ES and group the results by the level of the processing time variability (low or high), the number of jobs and the problem set. Each number is the average of the 10 problem instances per problem type.

In Table 3 and Table 4, we consider the performance of the heuristics individually, meaning that we compute the ratio of AESs found to MAX_ES without considering if any other heuristic found dominating AESs. We report the average values over the ten problem instances for KAK and Long BRKGA. For the five Short BRKGA runs, we report the average percentage as well as the best percentage over the five runs, averaged over the ten problem instances for each problem type.

Table 3: Percentage of AESs found, low variability case

			Long	Short BRKGA	
NumJobs	Prob Set	KAK	BRKGA	Avg	Best
50	I	20.14%	74.82%	53.53%	63.31%
	II	5.16%	54.19%	34.77%	40.65%
	III	14.32%	47.97%	10.31%	10.98%
	IV	5.72%	15.15%	6.68%	7.11%
	V	13.73%	54.94%	45.67%	57.94%
100	I	17.80%	49.00%	11.64%	12.20%
	II	7.10%	12.17%	4.79%	5.20%
	III	21.36%	6.13%	3.52%	4.06%
	IV	18.75%	65.63%	56.00%	66.88%
	V	6.91%	64.80%	37.57%	45.07%
150	I	4.99%	52.38%	10.98%	12.02%
	II	1.92%	17.28%	6.78%	7.36%
	III	3.08%	52.69%	37.85%	41.92%
	IV	11.32%	39.32%	9.87%	10.90%
	V	25.28%	16.71%	5.58%	6.08%
200	I	11.00%	10.17%	3.72%	4.07%
	II	4.08%	85.71%	69.39%	75.51%
	III	2.78%	65.74%	52.78%	61.11%
	IV	6.58%	57.89%	40.79%	65.79%
	V	2.94%	65.20%	35.39%	42.65%

Table 4: Percentage of AESs found, high variability case

			Long	Short BRKGA	
NumJobs	Prob Set	KAK	BRKGA	Avg	Best
50	I	8.54%	45.61%	14.10%	16.34%
	II	8.37%	8.06%	3.84%	4.24%
	III	4.00%	6.19%	2.14%	2.62%
	IV	1.08%	3.23%	1.35%	1.56%
	V	8.80%	19.41%	7.15%	7.54%
100	I	6.09%	4.99%	2.41%	2.79%
	II	10.45%	2.66%	1.03%	1.11%
	III	4.68%	1.91%	0.76%	0.82%
	IV	3.24%	47.06%	36.29%	48.82%
	V	3.42%	8.66%	4.01%	4.53%
150	I	3.16%	5.24%	2.27%	2.65%
	II	1.27%	3.65%	1.47%	1.71%
	III	5.95%	17.42%	7.72%	8.56%
	IV	7.46%	5.70%	2.37%	2.92%
	V	5.13%	3.77%	1.50%	1.83%
200	I	6.80%	2.28%	0.82%	1.00%
	II	2.13%	47.52%	45.82%	57.45%
	III	4.63%	53.40%	31.11%	37.96%
	IV	4.06%	49.89%	7.90%	9.26%
	V	1.84%	13.36%	5.34%	6.84%

KAK never found more than 40% of the MAX_ES potential efficient solutions, while Long BRKGA found over 90% of the MAX_ES potential efficient solutions for one particular problem. However, Long BRKGA had trouble with some problem instances, once finding only 1.49% of the MAX_ES potential efficient solutions, while KAK found 4.44% for that particular problem. Interestingly, the Best Short BRKGA run sometimes found more AES solutions than the Long BRKGA run, despite the much larger number of chromosomes evaluated in Long BRKGA.

The different number of AESs found and reflected in Table 3 and Table 4 does not indicate the quality of these solutions, however. For this reason, we combined all seven sets of AESs found for each problem instance and determined the non-dominated solutions in the combined list. We compute various figures based on this set and report them in Table 5 and Table 6. In the third column, we show the number of AESs in the superset of all AESs found, reported as a percentage of MAX_ES. The last three columns indicate the percentage of AESs in the superset attributable to KAK, the Long BRKGA run or any of the Short BRKGA runs; these three columns add up to 100% in each case.

Table 5: Percentage of non-dominated AESs found, low variability case

NumJobs	Prob Set	% found	KAK	Long BRKGA	Short BRKGA
50	I	65.72%	0.36%	34.39%	65.25%
	II	52.46%	10.25%	34.42%	55.33%
	III	43.59%	11.15%	88.85%	0.00%
	IV	19.05%	25.72%	74.28%	0.00%
	V	61.90%	11.84%	27.04%	61.12%
100	I	48.50%	20.61%	79.14%	0.26%
	II	20.56%	56.19%	43.81%	0.00%
	III	17.52%	68.24%	31.76%	0.00%
	IV	65.69%	1.19%	25.97%	72.84%
	V	50.83%	5.19%	29.41%	65.39%
150	I	51.98%	10.94%	89.06%	0.00%
	II	20.33%	30.13%	69.87%	0.00%
	III	55.01%	7.82%	21.43%	70.75%
	IV	45.20%	16.56%	83.44%	0.00%
	V	22.40%	43.97%	56.03%	0.00%
200	I	17.67%	67.49%	32.51%	0.00%
	II	80.99%	0.00%	86.26%	13.74%
	III	60.21%	0.00%	43.21%	56.79%
	IV	50.25%	0.00%	40.14%	59.86%
	V	48.96%	2.73%	51.26%	46.01%

In Table 5 and Table 6, we often see that the Short BRKGA runs often find no solutions that ended up in the set of non-dominated AESs, despite having five runs in which those solutions could have been found. For the number "0" to appear in Table 5 and Table 6, an algorithm must have not found even one AES in the superset in any of the 10 files whose performance is summarized in one cell; KAK and Short BRKGA obviously fall into this category, while Long BRKGA sometimes found no AESs in the superset in some individual problem instances. In individual problem instances, KAK sometimes found up to 95% of the AESs in the superset while Long BRKGA and Short BRKGA sometimes found 100% of the AESs in the super set.

Table 6: Percentage of non-dominated AESs found, high variability case

NumJobs	Prob Set	% found	KAK	Long BRKGA	Short BRKGA
50	I	39.41%	0.32%	68.50%	31.18%
	II	11.81%	28.88%	71.12%	0.00%
	III	7.30%	56.96%	43.04%	0.00%
	IV	7.28%	67.09%	32.91%	0.00%
	V	23.91%	8.34%	91.15%	0.52%
100	I	10.35%	70.45%	29.55%	0.00%
	II	7.16%	72.55%	27.45%	0.00%
	III	7.42%	85.96%	14.04%	0.00%
	IV	39.83%	0.12%	87.40%	12.47%
	V	9.69%	22.58%	77.42%	0.00%
150	I	7.41%	55.04%	44.96%	0.00%
	II	4.78%	66.99%	33.01%	0.00%
	III	22.40%	4.29%	95.71%	0.00%
	IV	8.47%	59.13%	40.87%	0.00%
	V	8.50%	74.62%	25.38%	0.00%
200	I	6.85%	84.23%	15.77%	0.00%
	II	47.99%	0.00%	24.21%	75.79%
	III	35.69%	0.00%	30.09%	69.91%
	IV	34.68%	0.00%	100.00%	0.00%
	V	19.31%	0.60%	99.40%	0.00%

The running time of KAK was never more than 0.16 seconds. The running times of the Short BRKGA runs were very consistent in the same problem instance, as the stopping criteria was based primarily on the number of chromosomes evaluated; no problem resulted in MAX_ES AESs being found. The running time of Long BRKGA is significantly longer than the Short BRKGA runs, reflecting the stopping criteria being ten times more chromosome evaluations. When the processing times have low variability, the running times tend to be lower than when the processing times have high variability. However, this may be due to the fact that the mean processing time is also higher in the case of high variability in processing times.

Table 7: Average running times of BRKGA runs

		Processing Times Variability			
		Low		High	
NumJobs	Prob Set	Long BRKGA	Short BRKGA	Long BRKGA	Short BRKGA
50	I	86.27	10.70	118.46	16.02
	II	167.00	20.88	415.35	45.70
	III	369.81	54.80	726.32	75.39
	IV	688.22	79.78	1094.23	111.52
	V	86.40	10.63	168.00	23.27
100	I	231.45	36.46	489.00	52.75
	II	549.62	63.73	813.90	86.15
	III	881.63	91.31	1166.86	121.47
	IV	101.58	11.15	119.89	18.39
	V	225.11	20.94	448.56	48.97
150	I	382.97	54.10	726.98	76.59
	II	691.93	80.54	1029.87	107.05
	III	85.93	11.43	171.65	24.32
	IV	237.95	38.37	523.70	57.04
	V	524.77	63.34	811.46	87.73
200	I	857.21	91.29	1162.30	125.92
	II	103.59	14.31	100.75	12.62
	III	198.53	25.76	190.19	23.72
	IV	277.83	37.16	477.31	57.36
	V	374.30	48.27	722.31	80.26

6. CONCLUSIONS

In this paper, we have presented an adaptation of RKGA to bi-criteria optimization and investigated its performance on a specific single machine bi-criteria scheduling problem. The research presented here contributes to the multi-objective scheduling literature by adapting RKGA to multi-objective problems, most notably by using two types of elite reproduction which do not use the same type of measure of fitness. A deterministic heuristic from the literature for the problem, referred herein as KAK, was implemented for comparison to the GA-based approaches. Despite the claim that KAK would find all AESs, the GA-based approaches found AESs that dominated the KAK solutions. Nonetheless, KAK found some solutions that remained in the superset of non-dominated AESs, considering all the AESs found by the algorithms. Clearly, in this problem, there is value in deterministic and meta-heuristics for this problem. Further research will be conducted to investigate whether applying a GA to the epsilon-constraint version of the problem is effective, following the approach of [6] in which simulated annealing is performed for each potential value of the maximum earliness to minimize total flowtime, as well as introducing the KAK solutions as initial members of the population in the GA-based approach.

7. REFERENCES

[1] Bean, J.C., Genetic algorithms and random keys for sequencing and optimization, *ORSA Journal on Computing*, 6, 1994, 154-160.

[2] Coello Coello, C.A. and Mariano Romero, C.E. Multicriteria scheduling problems, In *Multiple criteria optimization: state of the art annotated bibliography surveys*, Eds. M. Ehrgott and X. Gandibleux. Kluwer, Boston, 2002, 445-491.

[3] Goldberg, D.E., *Genetic algorithms in search, optimization, and machine learning*, Addison-Wesley, Reading, MA. 1989.

[4] Halliburton, R., *Introduction to Population Genetics*, Pearson Prentice Hall, Upper Saddle River, NJ. 2004.

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

[6] Karasakal, E.K. and Koksalan, M.,A simulated annealing approach to bicriteria scheduling problems on a single machine, *Journal of Heuristics*, 6(3), 2000, 311-327.

[7] Koksalan, M., Azizoglu, M., and Kondakci, S.K., Minimizing flowtime and maximum earliness on a single machine, *IIE Transactions*, 30(2) 1998, 192-200.

[8] Koksalan, M.M., A heuristic approach to bicriteria scheduling, *Naval Research Logistics*, 46(7), 1999, 777-789.

[9] Kurz, M.E. and Askin, R.G., Heuristic Scheduling of Parallel Machines with Sequence-dependent Setup Times, *International Journal of Production Research*, 39(16), 2001, 3747-3769.

[10] Kurz, M.E. and Askin, R.G., Scheduling flexible flow lines with sequence-dependent setup times, *European Journal of Operational Research*, 159(1), 2004, 66-82.

[11] Michalewicz, Z., *Genetic algorithms + data structures = evolution programs*, Springer-Verlag, Berlin. 1992.

[12] Norman, B.A. and Bean, J.C., A genetic algorithm methodology for complex scheduling problems, *Naval Research Logistics*, 46, 1999, 199-211.

[13] Rangsaritratsamee, R., Ferrell, W., and Kurz, M., Dynamic rescheduling using a bicriteria objective with genetic local search, *Computers and Industrial Engineering*, 46(1), 2004, 1-15.

[14] T'Kindt, V. and Billaut, J.-C., Multicriteria scheduling problems, In *Multiple criteria optimization: state of the art annotated bibliography surveys*, Eds. M. Ehrgott and X. Gandibleux. Kluwer, Boston, 2002, 445-491.

[15] Wang, C-S. and Uzsoy, R., A genetic algorithm to minimize maximum lateness on a batch processing machine, *Computers & Operations Research*, 29, 2002, 1621-1640.