Computers & Operations Research 36 (2009) 2707 – 2715

Contents lists available at ScienceDirect

Computers & Operations Research

journal homepage: www.elsevier.com/locate/cor

A genetic algorithm approach for the single machine scheduling problem with linear earliness and quadratic tardiness penalties

Jorge M.S. Valente*, José Fernando Gonçalves

LIAAD-INESC Porto L.A., Faculdade de Economia, Universidade do Porto, Rua Dr. Roberto Frias, 4200-464 Porto, Portugal

ARTICLE INFO

Available online 9 December 2008

Keywords:
Scheduling
Single machine
Linear earliness
Quadratic tardiness
Genetic algorithms

ABSTRACT

In this paper, we consider the single machine scheduling problem with linear earliness and quadratic tardiness costs, and no machine idle time. We propose a genetic approach based on a random key alphabet. Several genetic algorithms based on this approach are presented. These versions differ on the generation of the initial population, as well as on the use of local search. The proposed procedures are compared with existing heuristics, as well as with optimal solutions for the smaller instance sizes.
The computational results show that the performance of the proposed genetic approach is improved by the addition of a local search procedure, as well as by the insertion of simple heuristic solutions in the initial population. Indeed, the genetic versions that include either or both of these features not only provide significantly better results, but are also much faster. The genetic versions that use local search are clearly superior to the existing heuristics, and the improvement in performance over the best existing procedure increases with both the size and difficulty of the instances. These genetic procedures are also quite close to the optimum, and provided an optimal solution for most of the test instances.

Scope and purpose

This paper considers a single machine scheduling problem with linear earliness and quadratic tardiness costs, and no machine idle time. Scheduling with early and tardy penalties has received considerable attention from the scheduling community, due to its practical importance. Indeed, early/tardy scheduling problems are compatible with the concepts of Just-in-Time production and supply chain management, which have been adopted by many organizations.
Single machine scheduling environments actually occur in several practical applications. Also, the performance of many production systems is often determined by the schedules for a single bottleneck machine. Furthermore, the study of single machine problems frequently provides results that prove useful for more complex scheduling environments. The assumption that no machine idle time is allowed is also appropriate for many production settings. In fact, idle time should be avoided when the machine has limited capacity or high operating costs, and when starting a new production run involves high set-up costs or times.
In this paper, we present several algorithms based on a genetic approach that uses a random key alphabet. The various versions of the genetic algorithm differ on the generation of the initial population, as well as on the use of local search. These procedures are compared with existing heuristics, as well as with optimal solutions for some instance sizes.
The computational results show that inserting solutions generated by simple heuristics in the initial population, and using a local search procedure, enhances the performance of the proposed genetic approach. In fact, the addition of one or both of these features improves both the solution quality and the speed of the genetic algorithm. The genetic versions that apply local search clearly outperform the existing heuristics, and are quite close to the optimum solutions. Also, the improvement over the best existing procedure increases with both the size and the difficulty of the test instances.

* Corresponding author. Fax: +351 22 550 50 50.
E-mail address: jvalente@fep.up.pt (J.M.S. Valente).

0305-0548/$ - see front matter © 2008 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2008.11.016

1. Introduction

In this paper, we consider a single machine scheduling problem with linear earliness and quadratic tardiness costs, and no machine idle time. Single machine scheduling environments actually occur in several practical operations (for a recent example in the chemical industry, see [1]). Also, the performance of many production systems is frequently determined by the quality of the schedules for a single bottleneck machine. Moreover, results and insights obtained for single machine problems can often be applied to more complex scheduling environments, such as flow shops or job shops.

Earliness/tardiness scheduling models have received considerable and increasing attention from the scheduling community, due to their practical importance and relevance. In fact, scheduling problems with both earliness and tardiness penalties are compatible with the concepts of Just-in-Time production and supply chain management. These production strategies, which have been adopted by many organizations, view both early and tardy deliveries as undesirable.

We consider an objective function with linear earliness and quadratic tardiness costs. On the one hand, early completions of jobs result in unnecessary inventory. A linear penalty is then used for the early jobs, since the costs of maintaining and managing this inventory tend to be proportional to the quantity held in stock. On the other hand, late deliveries can result in lost sales, loss of goodwill, and disruptions in stages further down the supply chain. A quadratic tardiness penalty is appropriate in several practical settings, and is then used for the tardy jobs. Indeed, the tardiness is an important attribute of service quality, and a customer's dissatisfaction tends to increase quadratically with the tardiness, as proposed in the loss function of Taguchi [2]. Also, a quadratic tardiness penalty can in some situations be preferable to the more usual linear tardiness or maximum tardiness functions, as discussed in [3].

The assumption that no machine idle time is allowed is also appropriate for many production settings. In fact, idle time should be avoided when the machine has limited capacity or high operating costs. This assumption is also justified when starting a new production run involves high set-up costs or times. Some specific examples of production settings where the no idle time assumption is appropriate have been given by Korman [4] and Landis [5].

Formally, the problem we consider can be stated as follows. A set of n independent jobs $\{J_1, J_2, \ldots, J_n\}$ has to be scheduled on a single machine that can handle at most one job at a time. The machine is assumed to be continuously available from time zero onwards, and preemptions are not allowed. Job J_j, $j = 1, 2, \ldots, n$, requires a processing time p_j and should ideally be completed on its due date d_j. For a given schedule, the earliness and tardiness of J_j are, respectively, defined as $E_j = \max\{0, d_j - C_j\}$ and $T_j = \max\{0, C_j - d_j\}$, where C_j is the completion time of J_j. The objective is, then, to find a schedule that minimizes the sum of linear earliness and quadratic tardiness costs $\sum_{j=1}^{n} (E_j + T_j^2)$, subject to the constraint that no machine idle time is allowed.

This problem has been previously considered, and both exact and heuristic approaches have been proposed. Valente [6] developed both a lower bounding procedure based on a relaxation of the job completion times and a branch-and-bound algorithm. Among the heuristics, both dispatching rules [7] and beam search heuristics [8] have been proposed. The corresponding problem with inserted idle time was studied by Schaller [9], who presented a timetabling procedure, as well as a branch-and-bound algorithm and simple and efficient heuristics. Some problems with related objective functions have also been considered. These include the minimization of the linear earliness and tardiness costs $\sum_{j=1}^{n} (E_j + T_j)$ [10–12], and the minimization of the quadratic lateness [13–16], where the lateness of J_j is defined as $L_j = C_j - d_j$.

A large number of papers have been published on scheduling models with earliness and tardiness costs. Baker and Scudder [17] provide an excellent survey of the initial work on early/tardy scheduling. A recent survey of multicriteria scheduling problems is given by Hoogeveen [18]. This survey considers and reviews problems with earliness and tardiness penalties. Also, Kanet and Sridharan [19] give a review of scheduling models with inserted idle time that complements our focus on a problem with no machine idle time.

In this paper, we present several genetic algorithms, and analyse their performance on a wide range of instances. The proposed genetic approach uses a random key alphabet, so each chromosome is encoded as a vector of random numbers. The various versions of the genetic approach differ on the generation of the initial population, as well as on the use of local search. The genetic algorithms are compared with existing heuristics, as well as with optimal solutions for some instance sizes.

The remainder of this paper is organized as follows. In Section 2, we describe the proposed genetic algorithm approach, and present the several versions that were considered. The computational results are reported in Section 3. Finally, we provide some concluding remarks in Section 4.

2. The genetic algorithm procedures

In this section, we begin by briefly describing the main features of genetic algorithms. The encoding used to represent the problem solutions is then presented. The evolutionary strategy, i.e. the transitional process between consecutive populations, is also described. Finally, we present the four different versions that were considered for the proposed approach.

2.1. Genetic algorithms

Genetic algorithms are adaptive methods that can be used to solve optimization problems. The term genetic algorithm was first used by Holland [20], whose book *Adaptation in Natural and Artificial Systems*, published in 1975, was pivotal to the creation of what is now a large and quite active field of research. Actually, optimization played only a small part in Holland's work, even though it has since been the focus of the majority of the research on genetic algorithms. The literature on genetic algorithms includes a quite large number of papers; for some references describing in detail the genetic algorithm approach and its applications see [21–23].

Genetic algorithms are based on the evolution process that occurs in natural biology. Over many generations, natural populations tend to evolve according to the principles of natural selection or survival of the fittest, as first stated by Charles Darwin in *The Origin of the Species*. Genetic algorithms mimic this process, evolving populations of solutions to real world problems.

In order to apply a genetic algorithm to a specific problem, a suitable encoding or representation must first be devised. In this encoding, a solution to the problem is represented by a set of parameters. These parameters (known in genetic terminology as genes) are joined together in a string of values that represents or encodes the solution to the problem. In genetic terminology, this string is referred to as a chromosome or individual. A fitness value is associated with each chromosome. This value measures the quality or merit of the solution associated with that chromosome.

At each iteration, the genetic algorithm evolves the current population of chromosomes into a new population, using selection, crossover and mutation mechanisms. Some of the current individuals may be simply selected and copied to the new population. Additionally, the reproduction phase uses a crossover operator to combine individuals selected from the current population,

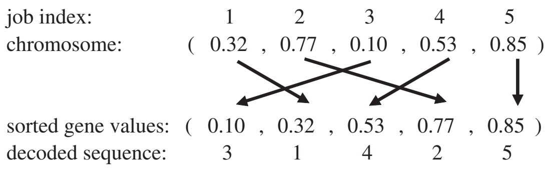
job index: 1 2 3 4 5
chromosome: (0.32 , 0.77 , 0.10 , 0.53 , 0.85)
sorted gene values: (0.10 , 0.32 , 0.53 , 0.77 , 0.85)
decoded sequence: 3 1 4 2 5

Fig. 1. Chromosome decoding example.

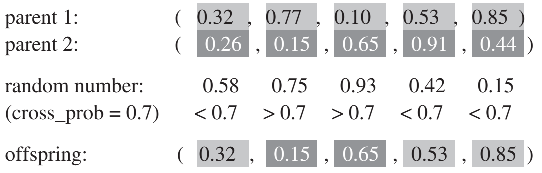
parent 1: (0.32 , 0.77 , 0.10 , 0.53 , 0.85)
parent 2: (0.26 , 0.15 , 0.65 , 0.91 , 0.44)
random number: 0.58 0.75 0.93 0.42 0.15
(cross_prob = 0.7) < 0.7 > 0.7 > 0.7 < 0.7 < 0.7
offspring: (0.32 , 0.15 , 0.65 , 0.53 , 0.85)

Fig. 2. Parameterized uniform crossover example.

producing offspring which are placed in the new population. The parent chromosomes are randomly selected from the current population, usually using a scheme which favours fitter individuals. The crossover operator then combines the genes of the two parents, yielding one or more offspring. Finally, a mutation operator is applied to some individuals, in order to change their genetic material (i.e. one or more of their genes).

The reproduction phase and the crossover operator tend to increase the quality of the populations. However, they also tend to force convergence of those populations. The mutation process can offset this convergence effect. Indeed, the mutation operator tries to guarantee the population diversity, and ensure an extensive search of the solution space.

2.2. Chromosome representation and decoding

The genetic algorithm approach proposed in this paper uses a random key alphabet $U(0,1)$ [24] to encode the chromosomes. In this alphabet, each gene is a uniform random number between 0 and 1. Therefore, a chromosome is encoded as a vector of random keys (random numbers). In our algorithms, each chromosome is made of n genes g_i, so the size of each chromosome is equal to the number of jobs: chromosome=$(g_1, g_2, \ldots, g_n)$.

In order to evaluate the fitness of an individual, it is necessary to decode its chromosome into the corresponding solution to the problem, i.e. into a sequence of the jobs. The decoding or mapping of a chromosome into a sequence is accomplished by sorting the jobs. The priorities used in this sorting operation are given by the genes. More specifically, the sorting priority of job J_i is equal to g_i (see Fig. 1 for an example).

An important feature of the random key alphabet is the fact that all offspring generated by crossover are feasible solutions. This is accomplished by moving the feasibility issue into the chromosome decoding procedure. If any vector of random numbers can be decoded into a feasible solution, then any chromosome obtained via crossover also corresponds to a feasible solution. Through its internal dynamics, the genetic algorithm then learns the relationship between random key vectors and solutions with good fitness and objective function values.

This feature is a significant advantage of the random key alphabet over the more natural encoding where each chromosome is a permutation of the job indexes. Indeed, with the natural encoding, the crossover operation is made more difficult and complicated by the need to assure that the resulting offspring correspond to a feasible solution.

2.3. Evolutionary strategy

A great number of genetic algorithm variants can be obtained by varying the selection, reproduction, crossover and mutation operators. We now describe the evolutionary strategy used in the proposed approach, i.e. the mechanisms that are used to generate a new population from the current set of individuals. The size of the population is kept constant throughout the procedure. This size is set as a multiple *pop_mult* of the size of the problem (i.e. the number of jobs n), where pop_mult is a user-defined parameter. This strategy has proved appropriate in our previous experience with genetic algorithms based on the same evolutionary approach [25–27].

Given a current population, the next population is obtained through elitist selection, crossover and migration mechanisms. We defer the discussion of the generation of the initial population to the next section. The calculation of the fitness value will also be addressed in that section.

The elitist selection strategy [21] consists in copying some of the best individuals in the current population to the new population. The number of individuals that are copied in the elitist selection phase is equal to a proportion *elit_prop* of the population size, where elit_prop is a user-defined parameter.

The advantage of the elitist selection strategy over the traditional approach where the entire population is completely replaced with new chromosomes is that the best individual in the population is monotonically improving over time. A potential downside is population convergence to a local minimum. This, however, can be overcome by high mutation or migration rates.

The migration mechanism replaces the traditional gene-by-gene mutation operator. In the migration phase, new individuals are randomly generated and added to the new population. The number of new randomly generated individuals is equal to a proportion *mig_prop* of the population size, where mig_prop is a user-defined parameter. As previously mentioned, the purpose of the migration mechanism is to prevent premature convergence and to assure the diversity of the population (like in the traditional mutation operator). This phase also guarantees that, if allowed to run for a sufficient amount of time, the proposed genetic approach will visit all possible solutions (and, therefore, also an optimal one).

The remaining individuals of the new population are generated via crossover. In the reproduction and crossover phase, two parents are initially selected. The first parent is randomly chosen from the elite individuals in the current population, i.e. the individuals that are copied to the new population in the elitist selection phase. The second parent is randomly selected from the full current population. The parameterized uniform crossover method [28], described below, is then used to obtain an offspring that is added to the new population. This process is repeated until the new population has been fully generated.

In the parameterized uniform crossover method, a random uniform number between 0 and 1 is generated for each gene. This random number is then compared with the user-defined parameter *cross_prob*. If the random number is less than or equal to cross_prob, the gene in the offspring is set equal to the corresponding gene in the first parent; otherwise, the value of the gene is copied from the second parent (see Fig. 2 for an example).

This evolutionary strategy is repeated until a stopping criterion is met. In our approach, the number of iterations without improvement stopping criterion was chosen. Therefore, the algorithms terminate when *stop_iter* populations have been generated without improving the best solution found so far, where stop_iter is a user-defined parameter. Fig. 3 depicts the evolutionary strategy, while the main steps of the proposed approach are given in Fig. 4.

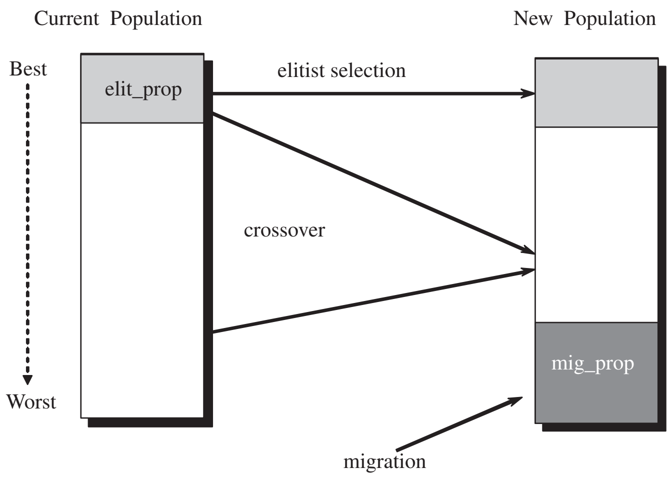

Fig. 3. Evolutionary strategy.

```
Genetic approach
{
        Generate Initial Population $\mathbf{P_t} = \mathbf{P_0}$;
        Evaluate Population $\mathbf{P_0}$;
        Update Best Solution;
        Set iter_no_improv = 0;

        While (iter_no_improv < stop_iter)
        {
                Generate New Population $\mathbf{P_{t+1}}$
                {
                        Perform Elitist Selection;
                        Perform Migration;
                        Perform Crossover;
                }

                Evaluate $\mathbf{P_{t+1}}$;

                If (new best solution is found)
                {
                        Update Best Solution;
                        Set iter_no_improv = 0;
                }
                Else
                        Set iter_no_improv = iter_no_improv + 1;

                Set $\mathbf{P_t} = \mathbf{P_{t+1}}$;
        }
}
```

Fig. 4. Genetic approach.

2.4. *Genetic algorithm versions*

The discussion of both the generation of the initial population and the calculation of the fitness value has been deferred to this section. In fact, two different strategies were used for each of these two issues. Therefore, we considered four genetic algorithm versions, corresponding to the various combinations of these strategies.

In the version denoted as GA, on the one hand, the initial population is randomly generated. The fitness value of a chromosome, on the other hand, is set equal to the symmetric of the objective function value of the corresponding sequence (i.e. the sequence obtained by decoding the chromosome, as previously described).

The GA_IN version differs from GA only in the initial population, which is not fully randomly generated. In the GA_IN version, we first introduce in the initial population three non-random chromosomes. Indeed, previous studies (e.g. [29,30]) have shown that this can improve the performance of a genetic algorithm. In this paper, this will be referred to as initializing the first population (hence the "_IN" part of the heuristic identifier).

The three non-random chromosomes are created so that their corresponding sequences are equal to the schedules generated by the EDD, SPT_s_j and EQTP_EXP dispatching heuristics analysed in [7]. The EDD (SPT_s_j) heuristic performed well for instances where most jobs are early (tardy). The EQTP_EXP dispatching rule, henceforth denoted simply by EQTP, was the best-performing of the heuristics considered in [7].

The MA and MA_IN versions differ from their GA and GA_IN counterparts only in the calculation of the fitness value. Indeed, these two versions additionally use a local search procedure in order to improve the decoded sequence. More precisely, in order to calculate the fitness of an chromosome, we first decode its corresponding sequence. A local search procedure is then used to improve this sequence. The fitness is set equal to the symmetric of the objective function value of the improved sequence. Finally, the chromosome is changed (i.e. its genes are rearranged) so that it corresponds to the improved sequence obtained after the application of the local search procedure. Since these two versions of the proposed genetic approach combine a genetic evolutionary strategy with a local search procedure, they can also be viewed as memetic algorithms [31] (hence the "MA" part of their identifiers).

We considered three simple local search procedures: adjacent pairwise interchange (API), 3-swaps (3SW) and largest cost insertion (LCI). The API procedure, at each iteration, considers in succession all adjacent job positions. A pair of adjacent jobs is then swapped if such an interchange improves the objective function value. This process is repeated until no improvement is found in a complete iteration. The 3SW procedure is similar, but it considers three consecutive job positions instead of an adjacent pair of jobs. All possible permutations of these three jobs are then analysed, and the best configuration is selected. Once more, the procedure is applied repeatedly until no improvement is possible. The LCI method selects at each iteration the job with the largest objective function value. The selected job is then removed from its position i in the schedule, and inserted at position j, for all $j \neq i$. The best insertion is then performed if it improves the objective function value. This process is repeated until no improving move is found.

3. Computational results

In this section, we first present the set of problems used in the computational tests, and specify the values that were used for the various parameters required by the genetic algorithms. The proposed genetic versions are then compared with existing heuristics, namely the EQTP dispatching rule developed in [7] and the recovering beam search (RBS) procedure presented in [8]. Finally, the heuristic results are evaluated against the optimum objective function values for some instance sizes. The instances used in the computational tests are available online at http://www.fep.up.pt/docentes/jvalente/benchmarks.html. The objective function value provided by the MA_IN, RBS and EQTP procedures, as well as the optimum objective function value (when available), can also be obtained at this address. For the MA_IN algorithm, and for each instance, the best, average and worst objective function values are provided.

3.1. *Experimental design and preliminary tests*

The computational tests were performed on a set of problems with 10, 15, 20, 25, 30, 40, 50, 75 and 100 jobs. These problems were randomly generated as follows. For each job J_j, an integer processing

Table 1
Parameter values.

	GA	GA_IN	MA	MA_IN
pop_mult	3	2	1	1
elit_prop	0.05	0.05	0.05	0.05
mig_prop	0.25	0.25	0.25	0.25
cross_prob	0.7	0.7	0.7	0.7
stop_iter	50	30	10	10
local search	–	–	API	API

time p_j was generated from one of the two uniform distributions [45, 55] and [1, 100], in order to obtain low (L) and high (H) variability, respectively, for the processing time values. For each job J_j, an integer due date d_j was generated from the uniform distribution $[P(1-T-R/2), P(1-T+R/2)]$, where P is the sum of the processing times of all jobs, T is the tardiness factor, set at 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0, and R is the range of due dates, set at 0.2, 0.4, 0.6 and 0.8.

For each combination of problem size n, processing time variability (var), T and R, 50 instances were randomly generated. Therefore, a total of 1200 instances were generated for each combination of problem size and processing time variability. All the algorithms were coded in Visual C++ 6.0, and executed on a Pentium IV-2.8 GHz personal computer. Due to the large computational times that would be required, the GA heuristic was not applied to the instances with 100 jobs.

We performed preliminary experiments to determine adequate values for the parameters required by the genetic algorithms. A separate problem set was used to conduct these preliminary experiments. This test set included instances with 25 and 50 jobs, and contained five instances for each combination of instance size, processing time variability, T and R. The instances in this smaller test set were generated randomly just as previously described for the full problem set. We considered the following values for the several parameters required by the proposed genetic algorithms:

pop_mult={1, 2, 3};
elit_prop={0.05, 0.10, 0.15, 0.20};
mig_prop={0.10, 0.15, 0.20, 0.25};
cross_prob={0.6, 0.7, 0.8};
stop_iter={10, 30, 50}.

The intervals for the elit_prop, mig_prop and cross_prob values were based on our previous experience with genetic algorithms based on the same evolutionary approach [25–27]. Indeed, we have consistently obtained good results with values inside the considered ranges. The intervals for the pop_mult and stop_iter parameters were determined after some initial testing. For the MA and MA_IN versions, we additionally considered the API, 3SW and LCI local search procedures, as previously mentioned.

The genetic algorithms were then applied to the test instances for all parameter (and local search procedure, for the MA and MA_IN versions) combinations. A thorough analysis of the objective function values and runtimes was then conducted, in order to select the values that provided the best trade-off between solution quality and computation time. The parameter values and local search procedure selected for the several genetic versions are given in Table 1.

The same elit_prop, mig_prop and cross_prob values proved adequate for all the versions. For the more sophisticated MA and MA_IN versions, the results were actually virtually identical for all the combinations of these parameters. For the GA and GA_IN versions, there were some small differences in performance, and the chosen values provided good results across all instance types.

Table 1 shows that smaller values are required for the parameters pop_mult and stop_iter as the genetic version becomes more sophisticated. Indeed, as local search and/or population initialization are introduced, smaller populations and/or a lower number of iterations without improvement can be used without compromising the solution quality.

Table 2
Comparison with the RBS heuristic–relative improvement.

n	Heur.	Low var.				High var.			
		%imp			all_equal	%imp			all_equal
		Best	Avg.	Worst		Best	Avg.	Worst	
10	GA	0.02	−0.18	−1.58	18.83	0.36	−0.31	−2.42	8.67
	GA_IN	0.02	−0.01	−0.10	47.83	0.35	−0.17	−1.48	24.00
	MA	0.02	0.01	−0.09	96.67	0.37	0.35	0.28	94.67
	MA_IN	0.02	0.02	0.02	99.17	0.37	0.37	0.35	98.17
20	GA	0.11	−0.26	−1.74	0.08	0.36	−0.96	−3.95	0.00
	GA_IN	0.10	0.05	−0.03	28.33	0.51	−0.17	−1.33	10.00
	MA	0.11	0.07	−0.13	74.50	0.66	0.50	0.02	62.33
	MA_IN	0.11	0.11	0.08	85.08	0.66	0.60	0.45	73.00
30	GA	0.22	−0.20	−1.42	0.00	0.35	−0.96	−4.07	0.00
	GA_IN	0.23	0.18	0.11	23.67	0.71	0.14	−0.67	5.33
	MA	0.25	0.21	0.03	67.00	0.98	0.76	0.23	47.17
	MA_IN	0.25	0.24	0.22	75.92	0.98	0.89	0.71	56.75
40	GA	0.43	−0.03	−1.49	0.00	0.59	−0.70	−3.40	0.00
	GA_IN	0.45	0.40	0.33	18.92	1.08	0.56	−0.18	3.33
	MA	0.48	0.44	0.37	63.58	1.45	1.23	0.81	41.08
	MA_IN	0.48	0.46	0.44	72.50	1.46	1.36	1.17	50.08
50	GA	0.42	−0.05	−1.63	0.00	0.68	−0.41	−2.48	0.00
	GA_IN	0.44	0.40	0.33	18.67	1.27	0.80	0.17	2.00
	MA	0.47	0.45	0.39	62.25	1.67	1.45	1.06	38.58
	MA_IN	0.47	0.46	0.44	70.83	1.69	1.59	1.42	45.58
75	GA	0.63	0.14	−1.15	0.00	1.00	−0.05	−2.01	0.00
	GA_IN	0.71	0.67	0.61	27.42	1.76	1.40	0.94	4.92
	MA	0.74	0.72	0.67	60.42	2.14	1.94	1.66	35.33
	MA_IN	0.74	0.73	0.72	66.83	2.18	2.09	1.96	41.08
100	GA	–	–	–	–	–	–	–	–
	GA_IN	0.76	0.73	0.68	29.00	2.31	2.01	1.62	5.00
	MA	0.80	0.78	0.76	58.58	2.73	2.57	2.35	32.50
	MA_IN	0.80	0.79	0.77	64.08	2.78	2.69	2.58	35.58

Finally, the API local search procedure was selected. This procedure provided results that were quite close to those given by the 3SW method, and was significantly faster. The LCI procedure was outperformed by the API and 3SW methods in both solution quality and computation time.

We recall that the parameter values were selected with the objective of obtaining the best trade-off between solution quality and computation time. Therefore, lower objective function values can still be obtained for some of the test instances, at the cost of increased computation times, by increasing the pop_mult or stop_iter values, or by selecting the 3SW local search procedure.

3.2. Comparison with existing heuristics

In this section, we compare the proposed genetic algorithms with existing heuristics, namely the RBS procedure and the EQTP dispatching rule. The EQTP heuristic provided the best results among all the dispatching rules analysed in [7]. The RBS procedure proposed in [8] currently provides the best results (in terms of solution quality) for the considered problem. For each instance, 10 independent runs were performed for all versions of the genetic algorithms.

Table 2 provides the mean relative improvement in objective function value over the RBS procedure (%imp), as well as the percentage number of times that the same objective function value is obtained for all of the 10 independent runs (all_equal). The relative improvement over the EQTP rule is given in Table 3. In Table 4, we give the percentage number of times each genetic version performs better (<), equal (=) or worse (>) than the EQTP and RBS procedures.

Table 3
Comparison with the EQTP heuristic–relative improvement.

n	Heur.	Low var.			High var.		
		Best	Avg.	Worst	Best	Avg.	Worst
10	GA	1.15	0.95	−0.42	9.54	9.00	7.25
	GA_IN	1.15	1.12	1.03	9.53	9.15	8.18
	MA	1.15	1.14	1.08	9.55	9.53	9.47
	MA_IN	1.15	1.15	1.15	9.55	9.55	9.53
20	GA	1.33	0.98	−0.46	6.30	5.25	2.89
	GA_IN	1.33	1.29	1.20	6.42	5.88	4.97
	MA	1.34	1.31	1.13	6.54	6.42	6.03
	MA_IN	1.34	1.34	1.31	6.54	6.50	6.39
30	GA	1.35	0.94	−0.25	5.15	4.05	1.54
	GA_IN	1.36	1.32	1.25	5.44	4.98	4.32
	MA	1.38	1.34	1.17	5.66	5.48	5.05
	MA_IN	1.38	1.37	1.35	5.66	5.59	5.45
40	GA	1.55	1.11	−0.31	4.89	3.80	1.54
	GA_IN	1.57	1.53	1.46	5.31	4.88	4.27
	MA	1.60	1.57	1.50	5.61	5.43	5.07
	MA_IN	1.60	1.59	1.57	5.62	5.54	5.39
50	GA	1.36	0.91	−0.62	4.53	3.61	1.90
	GA_IN	1.38	1.34	1.28	5.04	4.65	4.12
	MA	1.41	1.39	1.33	5.36	5.18	4.85
	MA_IN	1.41	1.40	1.39	5.38	5.30	5.17
75	GA	1.34	0.86	−0.40	4.07	3.17	1.50
	GA_IN	1.41	1.37	1.32	4.74	4.43	4.04
	MA	1.44	1.42	1.38	5.06	4.89	4.65
	MA_IN	1.45	1.44	1.42	5.09	5.02	4.91
100	GA	–	–	–	–	–	–
	GA_IN	1.30	1.27	1.22	4.94	4.67	4.34
	MA	1.33	1.32	1.30	5.29	5.16	4.96
	MA_IN	1.34	1.33	1.31	5.33	5.26	5.17

The relative improvement over the RBS and EQTP heuristics is calculated as (heur_ofv − ga_version_ofv)/heur_ofv × 100, where heur_ofv and ga_version_ofv are the objective function values of the corresponding heuristic (RBS or EQTP) and the appropriate genetic version, respectively. The avg. column provides the relative improvement calculated with the average of the objective function values obtained for all the 10 runs. In the best (worst) column, the relative improvement is calculated using the best (worst) result among the 10 runs.

The results in the avg. column provide an indication of the relative improvement we will obtain if the algorithm is executed only once, while the best column shows the improvement that can be achieved by performing 10 runs. The worst results are analogous to a lower bound on the performance of the genetic algorithms. Indeed, even when executed only once, the genetic algorithms will provide better results than those in the worst column, since no single seed gives the worst results for all instances.

The best performance is given by the MA_IN heuristic, closely followed by the MA algorithm. These two genetic versions are clearly superior to the best existing procedure, namely the RBS heuristic. Indeed, these algorithms provide an average relative improvement over the RBS procedure of about 2–3% (0.8%) for the largest high (low) variability instances. Also, these two heuristics give better results for a quite large percentage of the test instances, and are seldom inferior to the RBS procedure.

The GA_IN heuristic, on average, is somewhat outperformed by the RBS procedure for the smaller instance sizes. For the medium and large instances, however, the GA_IN algorithm is superior to the RBS heuristic. The GA procedure is the worst-performing of the genetic versions. When the best result over the 10 runs is considered, this heuristic does provide an improvement over the RBS procedure. However, the average results are usually inferior to those obtained with the RBS heuristic.

Table 4
Comparison with the EQTP and RBS heuristics—percentage of better, equal and worse results.

Heur.	Var.	n	GA			GA_IN			MA			MA_IN		
			<	=	>	<	=	>	<	=	>	<	=	>
EQTP	L	10	50.3	39.5	10.3	48.7	51.3	0.0	54.3	45.5	0.1	54.4	45.6	0.0
		20	48.7	11.9	39.4	59.0	41.0	0.0	71.0	28.2	0.8	71.8	28.2	0.0
		30	40.9	5.1	54.0	60.0	40.0	0.0	76.5	22.9	0.6	77.1	22.9	0.0
		40	37.7	2.9	59.4	62.7	37.3	0.0	82.6	17.2	0.3	82.8	17.2	0.0
		50	35.4	2.2	62.4	62.4	37.6	0.0	84.9	14.8	0.3	85.2	14.8	0.0
		75	33.7	1.2	65.1	60.2	39.8	0.0	78.7	21.1	0.3	78.9	21.1	0.0
		100	–	–	–	59.8	40.2	0.0	83.7	16.2	0.2	83.8	16.2	0.0
	H	10	73.5	17.6	8.9	71.5	28.5	0.0	77.7	22.2	0.2	77.7	22.3	0.0
		20	69.3	2.4	28.3	77.5	22.5	0.0	90.8	8.7	0.6	91.3	8.7	0.0
		30	63.0	0.5	36.5	78.0	22.0	0.0	95.6	3.9	0.4	96.1	4.0	0.0
		40	58.1	0.2	41.8	79.9	20.1	0.0	97.6	2.0	0.4	98.0	2.0	0.0
		50	56.2	0.1	43.8	79.5	20.6	0.0	98.9	0.8	0.4	99.3	0.8	0.0
		75	50.8	0.1	49.1	78.8	21.2	0.0	98.2	1.6	0.2	98.4	1.6	0.0
		100	–	–	–	76.3	23.7	0.0	99.5	0.3	0.1	99.7	0.3	0.0
RBS	L	10	2.1	79.7	18.1	2.1	78.4	19.4	2.9	96.8	0.3	3.0	96.9	0.1
		20	14.4	25.8	59.8	16.8	42.7	40.5	24.5	71.1	4.4	25.6	73.1	1.3
		30	17.1	9.0	74.0	27.4	31.0	41.6	39.9	56.8	3.3	41.9	57.4	0.7
		40	18.8	4.3	76.9	34.4	24.8	40.9	48.3	49.7	2.0	49.4	50.3	0.3
		50	16.3	2.9	80.8	34.9	22.7	42.4	51.0	47.2	1.8	52.4	47.4	0.2
		75	16.0	1.4	82.6	38.9	27.5	33.6	53.7	45.3	1.0	54.3	45.6	0.1
		100	–	–	–	40.0	24.8	35.3	57.4	41.9	0.7	57.8	42.2	0.0
	H	10	8.4	63.6	28.0	7.4	57.0	35.6	10.9	88.3	0.7	11.1	88.7	0.2
		20	17.7	10.5	71.8	18.5	17.3	64.2	38.6	54.4	7.0	40.8	56.2	3.1
		30	20.7	1.5	77.8	27.2	8.1	64.7	59.3	33.5	7.2	62.9	34.4	2.7
		40	22.7	0.2	77.2	37.9	5.4	56.6	72.5	22.2	5.3	76.0	22.7	1.4
		50	23.4	0.1	76.5	41.7	5.1	53.2	83.0	13.0	4.0	85.9	13.2	0.9
		75	24.1	0.1	75.7	50.6	8.1	41.2	90.3	7.9	1.8	91.8	8.1	0.1
		100	–	–	–	55.0	9.3	35.7	94.5	4.6	0.9	95.3	4.6	0.1

Table 5
Relative improvement over the RBS heuristic for instances with 50 jobs.

Heur.	T	Low var.				High var.			
		$R=0.2$	$R=0.4$	$R=0.6$	$R=0.8$	$R=0.2$	$R=0.4$	$R=0.6$	$R=0.8$
GA	0.0	−0.006	−0.013	−0.020	−0.019	−0.554	−0.700	−0.950	−1.130
	0.2	6.784	1.062	−0.037	0.131	11.251	−3.144	−4.203	−3.117
	0.4	−0.026	−0.112	−0.632	−7.333	0.405	−0.012	−0.493	−6.172
	0.6	−0.019	−0.055	−0.159	−0.451	0.202	0.016	−0.182	−0.622
	0.8	−0.014	−0.033	−0.049	−0.078	0.055	−0.056	−0.098	−0.127
	1.0	−0.008	−0.015	−0.021	−0.030	−0.030	−0.044	−0.052	−0.061
GA_IN	0.0	−0.001	−0.006	−0.011	−0.013	−0.090	−0.120	−0.153	−0.134
	0.2	6.801	1.522	0.177	0.258	11.361	2.408	1.955	1.615
	0.4	0.003	0.018	0.061	0.759	0.455	0.134	0.117	1.665
	0.6	0.000	−0.001	−0.002	−0.003	0.202	0.073	−0.024	−0.188
	0.8	−0.001	−0.001	0.000	−0.001	0.037	−0.024	−0.012	−0.006
	1.0	−0.001	0.000	0.000	0.000	−0.002	−0.010	−0.006	−0.003
MA	0.0	0.000	0.000	0.002	0.010	0.029	0.104	0.166	0.272
	0.2	6.867	1.649	0.292	0.430	12.032	4.003	3.389	3.887
	0.4	0.003	0.025	0.096	1.442	0.513	0.336	0.875	8.437
	0.6	0.002	0.001	0.001	0.000	0.257	0.153	0.121	0.064
	0.8	0.001	0.000	0.000	0.000	0.107	0.034	0.023	0.022
	1.0	0.000	0.000	0.000	0.000	0.001	0.002	0.004	0.003
MA_IN	0.0	0.000	0.001	0.004	0.013	0.054	0.130	0.214	0.322
	0.2	6.875	1.703	0.338	0.467	12.145	5.190	4.380	4.463
	0.4	0.007	0.026	0.097	1.538	0.562	0.381	0.893	8.634
	0.6	0.002	0.001	0.001	0.000	0.274	0.156	0.126	0.065
	0.8	0.001	0.000	0.000	0.000	0.107	0.034	0.023	0.022
	1.0	0.000	0.000	0.000	0.000	0.001	0.002	0.004	0.003

The genetic algorithms also considerably outperform the EQTP dispatching rule. The MA_IN and GA_IN versions necessarily always obtain a solution that is better than or equal to the EQTP schedule, since these procedures include the EQTP sequence in their initial population. The MA algorithm is also seldom inferior to the EQTP heuristic. Moreover, these three versions provide a relative improvement of over about 4–5% (1.3%) for the high (low) variability instances, even when the worst result over the 10 runs is considered. The EQTP rule is outperformed by even the GA procedure, the worst-performing of the genetic versions. Indeed, with the exception of the worst result for the low variability instances, the GA version provides a large relative improvement over the EQTP dispatching heuristic.

The best-performing MA_IN and MA versions are also quite robust. In fact, the best and worst results over the 10 runs are usually close. Also, the percentage of instances for which the same objective function value was obtained for all of the 10 runs is quite high for these two heuristics.

The performance of the genetic versions is improved by both the initialization of the first population, and the addition of a local search procedure. Indeed, the initialization of the first population improves the results, particularly when no local search is performed. Also, the use of a local search procedure leads to a substantial improvement, especially when the initial population is fully random. As previously mentioned, the best performance is achieved by the MA_IN version, which uses both initialization and local search.

The improvement given by the genetic algorithms over the existing heuristics is much larger when the processing time variability is high. Also, the improvement over the best existing procedure increases with the instance size. Indeed, the best-performing genetic versions provide an average improvement over the RBS heuristic of less than about 0.5% for the smallest instances with high variability. For the largest instances, however, this improvement is over 2.5%. Also, these genetic versions are quite rarely outperformed by the RBS procedure for the larger instances.

In Table 5, we present the effect of the, T and R, parameters on the relative improvement (calculated with the average objective function value) over the RBS procedure, for instances with 50 jobs.

Table 6
Runtimes (in seconds).

Var.	Heur.	$n=10$	$n=20$	$n=30$	$n=40$	$n=50$	$n=75$	$n=100$
L	EQTP	0.000	0.000	0.000	0.000	0.000	0.000	0.001
	RBS	0.001	0.004	0.009	0.019	0.033	0.100	0.227
	GA	0.012	0.074	0.221	0.492	0.932	3.028	–
	GA_IN	0.003	0.015	0.042	0.088	0.158	0.480	1.070
	MA	0.003	0.011	0.037	0.089	0.172	0.612	1.617
	MA_IN	0.004	0.011	0.033	0.079	0.153	0.545	1.445
H	EQTP	0.000	0.000	0.000	0.000	0.000	0.000	0.001
	RBS	0.001	0.004	0.009	0.020	0.035	0.109	0.250
	GA	0.013	0.080	0.242	0.549	1.051	3.476	–
	GA_IN	0.004	0.020	0.052	0.106	0.192	0.593	1.353
	MA	0.004	0.013	0.041	0.102	0.203	0.767	2.108
	MA_IN	0.004	0.011	0.036	0.089	0.172	0.649	1.790

The relative difference in objective function values is minor when most jobs are early ($T=0.0$), and when a larger number of jobs are tardy ($T \geqslant 0.6$). Actually, when most jobs are tardy ($T=1.0$), the objective function values are quite close for all the heuristics.

The relative difference in objective function values is much larger for the instances with a tardiness factor of 0.2 or 0.4. In fact, for these instances, the best-performing MA and MA_IN versions provide a relative improvement that can be as high as about 12% (6%) for the high (low) variability problems.

The heuristic runtimes (in seconds) are presented in Table 6; for the genetic versions, we provide the average runtime (i.e. the average of the runtimes for each of the 10 runs). As expected, the EQTP dispatching rule is quite clearly the most efficient of the heuristic procedures. The RBS procedure is faster than the genetic algorithms. Nevertheless, the GA_IN, MA and MA_IN versions are still efficient, since they are capable of solving instances with 100 jobs in less than about 2 s.

The GA procedure is significantly more computationally demanding than the other genetic versions. Indeed, the GA_IN, MA and MA_IN versions are much faster, even though they perform an initialization of the first population, and/or use a local search procedure. This is due to the lower values required for the pop_mult and stop_iter parameters, as previously mentioned. Therefore, the more sophisticated versions not only perform better, but are also faster, since they require smaller populations and/or a lower number of iterations without improvement.

The MA_IN version is then the recommended heuristic for small and medium instance sizes. This procedure provides the best results, and is relatively efficient. For large problems, however, a genetic algorithm approach will require excessive time. The RBS procedure can be applied to slightly larger instances than the genetic algorithms, but for the quite large problems only the EQTP dispatching heuristic will be able to provide results in reasonable computation times.

Table 7
Comparison with optimum objective function values.

Var.	Heur.	$n=10$		$n=15$		$n=20$	
		%dev	%opt	%dev	%opt	%dev	%opt
L	EQTP	1.78	45.58	2.14	34.50	1.83	28.17
	RBS	0.02	97.00	0.03	83.17	0.13	73.25
	GA	0.20	81.09	0.30	50.83	0.38	29.98
	GA_IN	0.03	79.59	0.05	56.53	0.06	44.23
	MA	0.01	99.56	0.02	97.31	0.04	91.75
	MA_IN	0.00	99.89	0.00	98.41	0.01	95.53
H	EQTP	22.14	22.25	16.45	11.92	11.96	8.67
	RBS	0.46	88.83	0.89	75.83	0.81	56.83
	GA	0.69	68.87	1.26	33.03	1.66	12.58
	GA_IN	0.55	60.18	0.89	28.88	0.86	17.48
	MA	0.02	98.98	0.09	93.80	0.16	84.37
	MA_IN	0.00	99.73	0.03	96.03	0.06	89.33

3.3. Comparison with optimum results

In this section, we compare the heuristic procedures with the optimum objective function values, for instances with up to 20 jobs. Table 7 gives the average of the relative deviations from the optimum (%dev), calculated as $(H-O)/O \times 100$, where H and O are the heuristic and the optimum objective function values, respectively. The percentage number of times each heuristic generates an optimum schedule (%opt) is also provided.

The MA_IN and MA versions perform extremely well. Indeed, the relative deviation from the optimum is quite small (usually less than 0.1%) for these heuristics. Also, these procedures provide an optimum solution for over 90% (and in some cases nearly all) of the test instances. The GA_IN, GA and RBS algorithms also provide good results, although inferior to those of the MA_IN and MA heuristics. The EQTP dispatching rule, on the other hand, is clearly outperformed by the other heuristic procedures.

The heuristics perform better when the processing time variability is low. This is particularly clear for the worst-performing

Table 8
Relative deviation from the optimum for instances with 20 jobs.

Heur.	T	Low var.				High var.			
		$R=0.2$	$R=0.4$	$R=0.6$	$R=0.8$	$R=0.2$	$R=0.4$	$R=0.6$	$R=0.8$
EQTP	0.0	0.188	0.087	0.080	0.111	0.661	1.641	2.655	2.640
	0.2	17.046	13.555	3.984	2.225	60.069	91.364	26.800	20.544
	0.4	0.099	0.127	0.424	5.923	6.336	4.565	13.489	44.532
	0.6	0.021	0.017	0.008	0.011	3.969	1.380	1.229	1.876
	0.8	0.012	0.007	0.003	0.002	1.677	0.820	0.298	0.247
	1.0	0.002	0.001	0.001	0.001	0.035	0.047	0.053	0.055
RBS	0.0	0.000	0.000	0.003	0.007	0.040	0.115	0.279	0.147
	0.2	1.857	0.208	0.227	0.266	3.974	4.516	2.308	1.423
	0.4	0.012	0.031	0.077	0.505	0.682	0.335	0.718	3.790
	0.6	0.003	0.002	0.000	0.000	0.673	0.133	0.057	0.045
	0.8	0.000	0.000	0.000	0.000	0.043	0.059	0.001	0.008
	1.0	0.000	0.000	0.000	0.000	0.000	0.000	0.001	0.000
GA	0.0	0.014	0.022	0.031	0.039	0.418	0.986	1.623	1.620
	0.2	0.073	0.256	0.397	0.458	1.013	5.955	7.442	6.676
	0.4	0.043	0.182	0.741	5.253	0.243	0.424	1.469	9.291
	0.6	0.031	0.106	0.236	0.737	0.148	0.256	0.451	0.853
	0.8	0.023	0.054	0.086	0.127	0.101	0.144	0.206	0.229
	1.0	0.013	0.025	0.037	0.058	0.050	0.082	0.086	0.107
GA_IN	0.0	0.010	0.009	0.019	0.022	0.191	0.317	0.515	0.529
	0.2	0.038	0.200	0.236	0.341	0.749	3.215	3.761	3.581
	0.4	0.011	0.020	0.057	0.460	0.240	0.390	1.058	4.114
	0.6	0.009	0.007	0.004	0.004	0.161	0.228	0.320	0.641
	0.8	0.004	0.003	0.002	0.001	0.094	0.175	0.134	0.135
	1.0	0.001	0.001	0.000	0.001	0.012	0.029	0.030	0.034
MA	0.0	0.001	0.004	0.004	0.008	0.025	0.075	0.129	0.114
	0.2	0.142	0.125	0.192	0.101	0.338	0.993	0.992	0.578
	0.4	0.029	0.045	0.013	0.325	0.108	0.086	0.114	0.267
	0.6	0.001	0.000	0.000	0.000	0.031	0.023	0.002	0.000
	0.8	0.000	0.000	0.000	0.000	0.002	0.001	0.000	0.000
	1.0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
MA_IN	0.0	0.000	0.000	0.001	0.000	0.015	0.027	0.026	0.030
	0.2	0.009	0.058	0.045	0.049	0.098	0.343	0.453	0.255
	0.4	0.000	0.001	0.003	0.038	0.026	0.015	0.036	0.145
	0.6	0.000	0.000	0.000	0.000	0.008	0.002	0.001	0.000
	0.8	0.000	0.000	0.000	0.000	0.002	0.000	0.000	0.001
	1.0	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000

heuristics (EQTP, RBS, GA and GA_IN). In fact, for low variability instances, the RBS procedure provides objective function values that are less than 0.2% above the optimum, and also generates an optimum solution for over 70% of the instances. Even the EQTP dispatching rule is only about 2% above the optimum. For instances with high variability, however, the performance deteriorates somewhat for the RBS algorithm, and quite considerably for the EQTP rule.

These results are in line with those presented in the previous section for the relative improvement provided by the genetic algorithms. Indeed, the relative improvement was lower (higher) for the instances with low (high) variability. This is in accordance with the results given in Table 7, since there is more room for improvement over the existing heuristics when the variability is high.

In Table 8, we present the effect of the T and R parameters on the relative deviation from the optimum, for instances with 20 jobs. The heuristics are closer to the optimum when most jobs are early ($T=0.0$), and when a larger number of jobs are completed after their due dates ($T \geqslant 0.6$). Indeed, when most jobs are tardy ($T = 1.0$), all the heuristic procedures are usually optimal or nearly optimal.

The relative deviation from the optimum is larger for the instances with $T=0.2$ or 0.4. The performance improvement provided by the MA_IN and MA heuristics is much clearer for these instances. In fact, for these instances, the relative deviation from the optimum for the RBS algorithm can be as high as about 4% (2%) for the instances with high (low) variability. For the MA_IN procedure, the relative deviation does not exceed 0.5% and 0.06% for the high and low variability instances, respectively. Again, these results are in line with those previously presented in Table 5.

4. Conclusion

In this paper, we presented a genetic approach for the single machine scheduling problem with linear earliness and quadratic tardiness costs, and no machine idle time. Several genetic algorithms based on this approach were presented. These versions differ on the generation of the initial population, as well as on the use of local search.

Initial experiments were first performed, in order to determine appropriate values for the parameters required by the genetic algorithms. The proposed procedures were then compared with existing heuristics, as well as with optimal solutions for the smaller instance sizes.

The MA_IN and MA genetic versions provided the best results, and are clearly superior to the existing procedures. These heuristics are also quite close to the optimum, and provided an optimal solution for over 90% (and in some cases nearly all) of the test instances. The improvement in performance provided by the genetic algorithms is larger for the more difficult instances, i.e. instances with high variability and/or a tardiness factor of 0.2 or 0.4. Also, the improvement over the best existing procedure increases with the instance size.

The performance of the proposed genetic approach was improved by both the initialization of the first population, and the addition of a local search procedure. Indeed, the MA_IN, MA and GA_IN versions not only provided much better results than the GA heuristic, but were also much faster. The additional time required by the population initialization and/or the local search procedure is more than offset by the fact that the more sophisticated versions require smaller populations and/or a lower number of iterations without improvement.

The MA_IN genetic algorithm is then recommended for small and medium instance sizes. Indeed, this procedure provided the best results, and can solve medium size problems within reasonable computation times. The RBS procedure can be applied to slightly larger instances than the genetic algorithms, but for the quite large problems only a dispatching heuristic, such as the EQTP rule, can provide results in reasonable computation times.

Acknowledgement

The authors would like to thank an anonymous referee for her/his most useful comments and suggestions that were used to improve this paper. This research was supported by Fundação para a Ciência e Tecnologia (FCT) project PTDC/GES/7224/2006.

References

[1] Wagner BJ, Davis DJ, Kher H. The production of several items in a single facility with linearly changing demand rates. Decision Sciences 2002;33:317–46.
[2] Taguchi G. Introduction to quality engineering. Tokyo, Japan: Asian Productivity Organization; 1986.
[3] Sun X, Noble JS, Klein CM. Single-machine scheduling with sequence dependent setup to minimize total weighted squared tardiness. IIE Transactions 1999;31:113–24.
[4] Korman K. A pressing matter. Video 1994; 46–50.
[5] Landis K. Group technology and cellular manufacturing in the Westvaco Los Angeles VH department. Project report in IOM 581, School of Business, University of Southern California; 1993.
[6] Valente JMS. An exact approach for the single machine scheduling problem with linear early and quadratic tardy penalties. Asia-Pacific Journal of Operational Research 2008;25:169–86.
[7] Valente JMS. Heuristics for the single machine scheduling problem with early and quadratic tardy penalties. European Journal of Industrial Engineering 2007;1:431–48.
[8] Valente JMS. Beam search heuristics for the single machine scheduling problem with linear earliness and quadratic tardiness costs. Working Paper 250, Faculdade de Economia, Universidade do Porto, Portugal; 2007. Asia-Pacific Journal of Operational Research, to appear.
[9] Schaller J. Single machine scheduling with early and quadratic tardy penalties. Computers & Industrial Engineering 2004;46:511–32.
[10] Garey MR, Tarjan RE, Wilfong GT. One-processor scheduling with symmetric earliness and tardiness penalties. Mathematics of Operations Research 1988;13:330–48.
[11] Kim YD, Yano CA. Minimizing mean tardiness and earliness in single-machine scheduling problems with unequal due dates. Naval Research Logistics 1994;41:913–33.
[12] Schaller J. A comparison of lower bounds for the single-machine early/tardy problem. Computers & Operations Research 2007;34:2279–92.
[13] Gupta SK, Sen T. Minimizing a quadratic function of job lateness on a single machine. Engineering Costs and Production Economics 1983;7:187–94.
[14] Sen T, Dileepan P, Lind MR. Minimizing a weighted quadratic function of job lateness in the single machine system. International Journal of Production Economics 1995;42:237–43.
[15] Su LH, Chang PC. A heuristic to minimize a quadratic function of job lateness on a single machine. International Journal of Production Economics 1998;55: 169–75.
[16] Schaller J. Minimizing the sum of squares lateness on a single machine. European Journal of Operational Research 2002;143:64–79.
[17] Baker KR, Scudder GD. Sequencing with earliness and tardiness penalties: a review. Operations Research 1990;38:22–36.
[18] Hoogeveen H. Multicriteria scheduling. European Journal of Operational Research 2005;167:592–623.
[19] Kanet JJ, Sridharan V. Scheduling with inserted idle time: problem taxonomy and literature review. Operations Research 2000;48:99–110.
[20] Holland JH. Adaptation in natural and artificial systems. Ann Arbor, Michigan: University of Michigan Press; 1975 (re-issued in 1992 by MIT Press).
[21] Goldberg DE. Genetic algorithms in search, optimization, and machine learning. Reading, MA: Addison-Wesley; 1989.
[22] Reeves CR. Genetic algorithms for the operations researcher. INFORMS Journal on Computing 1997;9:231–50.
[23] Reeves C. Genetic algorithms. In: Glover F, Kochenberger GA, editors. Handbook of metaheuristics. Dordrecht: Kluwer Academic Publishers; 2003. p. 55–82.
[24] Bean JC. Genetics and random keys for sequencing and optimization. ORSA Journal on Computing 1994;6:154–60.
[25] Gonçalves JF. A hybrid genetic algorithm-heuristic for a two-dimensional orthogonal packing problem. European Journal of Operational Research 2007;183:1212–29.
[26] Gonçalves JF, Mendes JJM, Resende MGC. A hybrid genetic algorithm for the job shop scheduling problem. European Journal of Operational Research 2005;167:77–95.
[27] Gonçalves JF, Resende MGC. An evolutionary algorithm for manufacturing cell formation. Computers & Industrial Engineering 2004;47:247–73.
[28] Spears WM, De Jong KA, On the virtues of parameterized uniform crossover. In: Belew R, Booker L, editor. Proceedings of the fourth international conference on genetic algorithms. San Mateo, CA: Morgan Kaufman; 1991. p. 230–236.
[29] Reeves CR. A genetic algorithm for flowshop sequencing. Computers & Operations Research 1995;22:5–13.
[30] Ahuja RK, Orlin JB. Developing fitter GAs. INFORMS Journal on Computing 1997;9:251–3.
[31] Moscato P, Cotta C. A gentle introduction to memetic algorithms. In: Glover F, Kochenberger GA, editors. Handbook of metaheuristics. Dordrecht: Kluwer Academic Publishers; 2003. p. 105–44.