Asia-Pacific Journal of Operational Research
Vol. 23, No. 3 (2006) 393–405

A HYBRID GENETIC ALGORITHM FOR THE EARLY/TARDY SCHEDULING PROBLEM

JORGE M. S. VALENTE*, JOSÉ FERNANDO GONÇALVES
and RUI A. F. S. ALVES

Faculdade de Economia da Universidade do Porto
Rua Dr. Roberto Frias, 4200-464 Porto, Portugal
**jvalente@fep.up.pt*

Received 3 September 2004
Revised 1 October 2005

In this paper, we present a hybrid genetic algorithm for a version of the early/tardy scheduling problem in which no unforced idle time may be inserted in a sequence. The chromosome representation of the problem is based on random keys. The genetic algorithm is used to establish the order in which the jobs are initially scheduled, and a local search procedure is subsequently applied to detect possible improvements. The approach is tested on a set of randomly generated problems and compared with existing efficient heuristic procedures based on dispatch rules and local search. The computational results show that this new approach, although requiring slightly longer computational times, is better than the previous algorithms in terms of solution quality.

Keywords: Scheduling; early/tardy; heuristics; genetic algorithms; random keys.

1. Introduction

In this paper, we consider a single machine scheduling problem with earliness and tardiness costs and no unforced machine idle time. Scheduling models with both early and tardy costs are compatible with the philosophy of just-in-time production, which emphasizes producing goods only when they are needed, since jobs are scheduled to complete as close as possible to their due dates. The early cost may represent the cost of completing a project early in PERT-CPM analyses, deterioration in the production of perishable goods or a holding cost for finished goods. The tardy cost can represent rush shipping costs, lost sales and loss of goodwill. It is assumed that no unforced machine idle time is allowed, and therefore the machine is only idle when no jobs are available for processing. This assumption represents a type of production environment where the machine idleness cost is higher than the cost incurred by completing a job early, or the machine is heavily loaded, so it must

*Corresponding author.

be kept running in order to satisfy the demand. Korman (1994) and Landis (1993) give some specific examples of production settings with these characteristics.

Formally, the problem considered in this paper can be stated as follows. A set of n independent jobs $\{J_1, J_2, \ldots, J_n\}$ has to be scheduled without preemptions on a single machine that can handle at most one job at a time. The machine and the jobs are assumed to be continuously available from time zero onwards and machine idle time is 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 any given schedule, the earliness and tardiness of J_j can be respectively defined as $E_j = \max\{0, d, -C_j\}$ 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 the schedule that minimizes the sum of the earliness and tardiness costs of all jobs $\sum_{j=1}^{n}(h_j E_j + w_j T_j)$, where h_j and w_j are the earliness and tardiness penalties of job J_j.

As a generalization of weighted tardiness scheduling (Lenstra *et al.*, 1977), the problem is strongly NP-hard. A large number of papers consider scheduling problems with both earliness and tardiness costs. We will only review those papers that examine a problem that is exactly the same as ours. For more information on earliness and tardiness scheduling, interested readers are referred to Baker and Scudder (1990), who provide an excellent review.

Ow and Morton (1989) developed several early/tardy dispatch rules and a filtered beam search procedure. Their computational studies show that the early/tardy dispatch rules, although clearly outperforming known heuristics that ignored the earliness costs, are still far from optimal. The filtered beam search procedure consistently provides very good solutions for small or medium size problems, but requires excessive computation times for larger problems (more than 100 jobs). Li (1997) presented a neighborhood search heuristic procedure that is superior to Ow and Morton's filtered beam search approach in terms of both efficiency and solution quality. Valente and Alves (2005a) presented additional dispatch and greedy heuristics, and also considered the use of improvement procedures to reduce the cost of the schedule obtained by the heuristics. Exact approaches have also been considered, and branch-and-bound algorithms were presented by Abdul-Razaq and Potts (1988), Li (1997) and Liaw (1999). The lower bounding procedure of Abdul-Razaq and Potts was based on the subgradient optimization approach and the dynamic programming state-space relaxation technique, while Li and Liaw used Lagrangean relaxation and the multiplier adjustment method. Valente and Alves (2005b) show that using better initial sequences can improve the lower bounds developed by Li and Liaw.

In this paper, some heuristic algorithms based on a combination of a genetic algorithm (GA) and local search procedures are presented. GAs have become a popular operational research tool and have been successfully used on several problems. Some examples of the application of GAs to quite different problems can be found in Bailey *et al.* (1997) (staff scheduling), Tan *et al.* (2001) (vehicle routing with time windows), Ahuja *et al.* (2000) (quadratic assignment) and Jaramillo *et al.*

(2002) (location problem). The proposed algorithms has been tested and compared to previous heuristic procedures. The computational results show that this new approach, although requiring slightly longer computing times, is better than the previous algorithms in terms of solution quality.

This paper is organized as follows. Section 2 describes the general characteristics of the GA such as its chromosome representation and the selection, mutation and crossover procedures. Section 3 presents several variations of the GA that emerged from its combination with some local search procedures. Computational results are presented and discussed in Section 4, and Section 5 contains the main conclusions.

2. Genetic Algorithm and Chromosome Representation

The GA uses a random key alphabet, where each key is a random number U(0,1), and an evolutionary strategy identical to the one proposed by Bean (1994). The important feature of random keys is that all offspring formed by crossover are feasible solutions, which is accomplished by moving the feasibility issue into the objective evaluation procedure.

A chromosome represents a solution to the problem and is encoded as a vector of random keys. Each chromosome solution is made of n genes, where n is the number of jobs. The genes are sorted in increasing order of their values to determine the order in which the jobs are going to be processed, and the fitness of each chromosome is set equal to the objective function value corresponding to the decoded processing order of the jobs.

The population of random key vectors is operated upon by the genetic algorithm reproduction, crossover, and mutation operators to breed good solutions. Reproduction is accomplished by copying the best individuals from one generation to the next, called an elitist strategy (Goldberg, 1989). The parameter *top* defines the proportion of the previous population best chromosomes that are copied to the next generation.

Parameterised uniform crossovers (Spears and DeJong, 1991) are employed in place of the traditional one-point or two-point crossover. After two parents are chosen randomly from the full old population (including chromosomes copied to the next generation in the elitist pass), at each gene a biased coin is tossed to select which parent will contribute the offspring. The probability of tossing a head is defined by the parameter *pcross*. In order to prevent premature convergence, and instead of using the traditional gene mutation, at each generation one or more new members of the population are randomly generated from the same distribution as the original population. The parameter *new* controls the proportion of the population chromosomes that are randomly generated.

The total number of individuals in the population is set as a multiple (defined by the parameter *nchrom*) of the number of jobs in the problem. The GA stops when a certain number of generations, controlled by the *maxger* parameter, is reached.

3. Combination of the GA with Local Search Procedures

This section presents a series of algorithms that resulted from a combination of the GA described in the previous section and some local search procedures. The most basic of all such versions, denoted by GA, consists of the GA without any type of local search, and will serve primarily as a benchmark to evaluate the performance of the other, more evolved, versions. Two other variants result from the use of the local search procedure at the fitness evaluation step of the GA. The first of these versions performs a single pass of a typical adjacent interchange procedure to attempt to improve the fitness of the chromosome, and will be denoted by genetic algorithm-single adjacent interchage (GA-SAI). The second variant is similar, but performs several consecutive passes of the adjacent interchange procedure, stopping when no improvement is made during a pass or when a limit of eight passes has been reached. This limit of eight passes was arrived at after some experimentation, during which it was also concluded that imposing no such limit (and stopping only when no further improvement could be made) would almost always increase the computational time without achieving relevant decreases in the fitness value. The experiments also showed that this limit value of eight passes was adequate for all instance sizes considered. This version will be denoted by genetic algorithm-multiple adjacent interchange (GA-MAI).

Three other algorithms result from the application of an extra local search procedure to each of the previously described versions. This new type of local search consists in performing successive passes of a non adjacent pairwise interchange method to the best solution obtained by the GA, stopping only when no further improvement is achieved in the objective function value. The non-adjacent pairwise interchange procedure iterates forward through each position in the sequence and considers the effect of swapping the job in that position with the jobs in all other positions (and not merely the adjacent ones). The best of those interchanges is then performed and this process is repeated for the next position in the sequence. These versions will be identified by appending multiple non adjacent interchange (MNAI) to the names of the previous versions.

In all the previously described versions, the GA is initialized solely with randomly generated chromosomes. The final six variations are different, because two of their initial chromosomes are not random but correspond to the solutions obtained by two other heuristics used for comparison purposes, which are presented below. These variants will be identified by the presence of the label INI, for initialisation.

We will, therefore, consider a total of 12 variants of the GA. As an example of the chosen identifiers, GA-SAI-MNAI refers to the version with no initialisation where a single pass of the adjacent interchange procedure is applied, and where a final round of multiple non adjacent interchange is performed, while GA-MAI-INI corresponds to the variant with initialisation where multiple passes of the adjacent interchange procedure are performed (but without the final non-adjacent local search method).

The GA was compared with two existing heuristic procedures, namely the EXP-ET dispatch rule presented by Ow and Morton (1989) and the NSearch neighborhood search algorithm proposed by Li (1997). The EXP-ET heuristic is one of the best performing dispatching procedures available for the early/tardy problem, while the NSearch algorithm outperforms all existing heuristics in terms of solution quality.

The EXP-ET dispatch rule uses the following priority index $P_j(t)$ to determine the job j to be scheduled at any instant t when the machine becomes available:

$$P_j(t) = \begin{cases} W_j, & \text{if } s_j \le 0 \\ W_j \exp\left(-\frac{H_j+W_j}{H_j}\left(\frac{s_j}{k\overline{p}}\right)\right), & \text{if } 0 \le s_j \le \left(\frac{W_j}{H_j+W_j}\right) k\overline{p} \\ H_j^{-2}\left(W_j - \frac{(H_j+W_j)s_j}{k\overline{p}}\right)^3, & \text{if } \left(\frac{W_1}{H_j+W_j}\right) k\overline{p} \le s_j \le k\overline{p} \\ -H_j, & \text{if } s_j \ge k\overline{p} \end{cases},$$

where $W_j = w_j/p_j,\ H_j = h_j/p_j,\ s_j = d_j - t - p_j$ is the slack of job j at time t, $\overline{p}$ is the average processing time of all jobs and k is an empirical integer lookahead parameter whose value must be specified. Whenever the machine is available and there are unscheduled jobs, the EXP-ET rule selects the job with the highest priority value.

The NSearch algorithm generates an initial schedule and then attempts to improve it using pairwise interchanges of jobs. The initial solution is determined as follows: Let J be the current partial schedule of m jobs and S the set of l unscheduled jobs; for each job $j \in S$, schedule j as the first job following J and then use a dispatch rule to schedule the remaining jobs in S following j. For each $j \in S$ one obtains a complete schedule and its total cost. The job with the lowest cost value is then selected as the next job in the sequence. The dispatch rule chosen to be used in this procedure was the EXP-ET heuristic. A neighborhood search is then conducted using a series of small neighborhood operators. Li defines OP_d as the pairwise interchange of two jobs with d jobs between them, $d = 0, 1, \ldots, n-2$. The initial operator is set as OP_o, which is the traditional adjacent pairwise interchange. The current operator is applied to the current sequence to generate a neighborhood and if the best schedule in that neighborhood is better than the current one, it becomes the new current sequence. If none of the schedules is better than the current sequence, the procedure stops if a stopping condition is met, otherwise we change to another operator by setting $d = d + 1$ if $d < k$ or $d = 0$ if $d = k$ (where k is the previously mentioned empirical lookahead parameter). The stopping condition terminates the algorithm when all the operators consecutively fail to improve the current schedule.

4. Computational Results

In this section, we present some computational results and compare the performance of the several versions of the GA and the two heuristic procedures described in

the previous section. The algorithms were tested on randomly generated problems with 15, 50, 75, and 100 jobs. The test problems were generated by a procedure also used by Abdul-Razaq and Potts (1988), Li (1997) and Liaw (1999). For each job j, an integer processing time p_j, earliness penalty h_j and tardiness penalty w_j are generated from the uniform distribution [1, 10]. An integer due date d_j is then generated, for each job j, from the uniform distribution $[T(1-LF-RDD/2), T(1-LF+RDD/2)]$, where T is the sum of the processing times of all jobs, LF is the lateness factor (set at 0.2 and 0.4), and RDD is the range of due dates (set at 0.2, 0.4, 0.6, 0.8, and 1.0). For each problem size, ten problems were generated for each combination of these two parameters, resulting in a total of 100 instances per problem size.

The GA parameters were set at the following values for all versions of the GA: $top = 10\%$; $new = 20\%$; $pcross = 70\%$ and $nchrom = 2$. These parameter values had proved effective in the authors' previous experiences with this GA in several other settings, and were thus chosen as a starting point. Nevertheless, and in order to confirm their effectiveness in the current context, it was decided to conduct a simple sensitivity analysis. The GEN algorithm was applied to a subset of the 50 job instances, and each of the starting parameter values was compared, in turn (i.e. keeping all other parameters at their initial settings), with two other neighboring values. The results of these tests are given in Table 1. This table gives both the average objective function value (AFV) and the average computation time (ACT), in seconds, for each parameter value. These results show that not only do the default values for the parameters prove to be reasonable choices, but also that the algorithm seems robust as far as the choice of the values for these parameters is concerned, since there are only minor deviations in performance for the different values that were tested.

The EXP-ET and NSearch heuristics use an empirical parameter k whose value must be specified. Three different values were considered for this parameter

Table 1. Sensitivity analysis results.

	$Top = 10\%$	$Top = 15\%$	$Top = 20\%$
AFV	8312	8451	8571
ACT	3.0	2.8	2.7
	$New = 15\%$	$New = 20\%$	$New = 25\%$
AFV	8472	8312	8414
ACT	2.9	3.0	2.9
	$Pcross = 60\%$	$Pcross = 70\%$	$Pcross = 80\%$
AFV	8481	8312	8378
ACT	2.9	3.0	2.9
	$Nchrom = 1$	$Nchrom = 2$	$Nchrom = 3$
AFV	9124	8312	8088
ACT	1.4	3.0	4.4

(3, 5, and 10), but for each problem size we will only present the results for the best of these values. The best parameter value for the EXP-ET dispatch rule was equal to 3 for all problem sizes. For the NSearch procedure, a value of $k = 3$ provided the best results for instances with 15 jobs, while $k = 5$ proved to be the best choice for the 50 job problems. For the remaining test problems, a value of $k = 10$ allowed for the best results. The results for the different parameters were close as far as the objective function value is concerned, but the versions with a higher value of k required a slightly longer run time. As for the versions of the GA in which initialization is performed, the two non random initial chromosomes correspond to the sequences generated by the EXP-ET heuristic with a lookahead of 3 and by the NSearch procedure with $k = 5$.

We now present the results of the computational tests. The objective function values obtained by the heuristic procedures are compared with the optimal solution for the 15 job problems, and with the best heuristic solution for the remaining problems. In Table 2 we present the average percent deviation (APD) from the optimum or the best heuristic solution (as appropriate), and the ACT, in seconds. Table 3 gives the number of instances for which the genetic versions perform better ($<$), equal ($=$) or worse ($>$) than the NSearch heuristic. The genetic algorithm results presented in these two tables were obtained by setting the *maxger* parameter at 500 (i.e. the genetic versions iterate until a limit of 500 generations is reached).

It can be seen that the EXP-ET dispatch rule is the fastest of all algorithms, but since it is the poorest in terms of solution quality we will henceforth focus on the results provided by the genetic versions and the NSearch procedure. As far as solution quality is concerned, the several variants of the GA (with the exception of GA, the simplest version, and the GA-INI and GA-SAI variants for the larger

Table 2. Average percent deviation and computation times (*maxger* = 500).

Algorithm	$n = 15$		$n = 50$		$n = 75$		$n = 100$	
	APD	ACT	APD	ACT	APD	ACT	APD	ACT
EXP-ET	18.6	0.0	15.1	0.0	12.8	0.0	14.5	0.0
NSearch	1.2	0.0	2.6	1.8	2.1	9.9	2.4	25.5
GA	1.9	1.7	6.4	15.2	7.1	39.4	9.5	75.4
GA-MNAI	0.7	1.7	1.2	15.5	0.8	40.7	0.7	78.8
GA-INI	0.8	2.0	1.9	16.7	2.2	44.0	2.5	91.8
GA-MNAI-INI	0.6	2.0	1.1	16.9	1.1	44.9	1.0	94.1
GA-SAI	0.0	2.3	1.3	21.9	2.1	63.4	3.0	114.5
GA-SAI-MNAI	0.0	2.3	0.5	22.1	0.7	64.4	0.9	117.1
GA-SAI-INI	0.0	2.8	0.6	26.7	1.0	74.5	1.1	148.1
GA-SAI-MNAI-INI	0.0	2.8	0.4	26.9	0.5	75.1	0.5	149.9
GA-MAI	0.0	3.6	0.2	46.6	0.5	130.0	0.8	267.7
GA-MAI-MNAI	0.0	3.6	0.2	46.7	0.3	130.6	0.4	269.6
GA-MAI-INI	0.0	3.9	0.2	50.8	0.4	130.5	0.5	260.2
GA-MAI-MNAI-INI	0.0	3.9	0.2	50.9	0.3	130.9	0.3	261.5

Table 3. Comparison of heuristic objective function values with NSearch results ($maxger = 500$).

Algorithm	$n = 15$			$n = 50$			$n = 75$			$n = 100$		
	$<$	$=$	$>$	$<$	$=$	$>$	$<$	$=$	$>$	$<$	$=$	$>$
GA	22	39	39	11	0	89	4	0	96	4	0	96
GA-MNAI	26	56	18	69	1	30	81	1	18	87	0	13
GA-INI	26	52	22	51	49	0	48	0	52	47	0	53
GA-MNAI-INI	27	56	17	69	31	0	71	0	29	86	0	14
GA-SAI	38	61	1	65	5	30	61	0	39	47	0	53
GA-SAI-MNAI	38	61	1	79	5	16	92	1	7	86	0	14
GA-SAI-INI	37	61	2	89	11	0	83	1	16	78	0	22
GA-SAI-MNAI-INI	37	61	2	91	9	0	88	1	11	90	0	10
GA-MAI	38	62	0	89	7	4	92	1	7	88	1	11
GA-MAI-MNAI	38	62	0	89	7	4	98	0	2	95	0	5
GA-MAI-INI	38	62	0	95	5	0	93	0	7	95	0	5
GA-MAI-MNAI-INI	38	62	0	95	5	0	93	0	7	97	0	3

instance sizes) are better than the results of the NSearch heuristic, both in average percent deviation and in the number of instances for which better results are obtained. The run time of the genetic versions are longer (particularly for the versions that incorporate more sophisticated local search procedures), but these times are for the full 500 generations the genetics iterate through.

Increased local search at the fitness evaluation level provides better solution values, as MAI is superior than SAI, which itself is superior to the simple GA. The run times increase as the local search complexity itself increases, but once again these results can be misleading, and need to be complemented by an analysis of the number of generations needed to reach the best solution. Including the final round of multiple non adjacent interchange is barely noticeable in terms of run time and can provide a further improvement in solution quality. However, this improvement is quite distinct in simpler and more evolved versions of the GA. For the simpler versions (GA with or without INI, SAI), the performance boost is quite visible, and the MNAI versions clearly win over their non-MNAI counterparts. For versions with increased local search at fitness evaluation level and/or initialization procedure (such as SAI-INI, MAI with and without INI) the performance of comparable MNAI and non-MNAI versions is quite similar. In fact, and particularly for the 50 and 75 job instances, the MNAI versions of these algorithms fail to improve on the best solution obtained by the genetic for most of the test problems. When the versions with and without INI are compared, it can be observed that not only the initialization procedure leads to better solution quality, but also that such improvement is much more visible in the simpler versions of the GA. In fact, the initialization procedure improves the solution quality for the GA and SAI versions, but leads to little improvement in the corresponding MNAI versions, as well as for the MAI with or without MNAI. As for the run time, the initialization versions require, as

expected, longer times. The exception is the MAI algorithm when tested on the 100 job instances, where the initialization versions are faster. This is probably due to savings in the local search performed at fitness evaluation, since improved solutions (through initialization) can lead to fewer passes of the adjacent interchange procedure.

The computation times presented for the GA are, as previously mentioned, for the full 500 generations. Therefore, one must also analyze the number of generations required for each version to reach its best solution, since it is possible that a version with a higher run time may require far less generations to reach its best value. Table 4 presents, for each version of the GA, results for the average value and for the percentiles 50 and 80 of the number of generations it took each of those versions to reach their best solution. The results show that, particularly for the versions with initialization, and for the MAI version (with and without INI), one can decrease the number of iterations, and consequently the run time, without having a negative impact on the solution quality for a large number of the instances.

The results in Table 4 show that, when there is local search at the fitness evaluation level, the algorithm converges to its best solution in less iterations. In fact, both for versions with and without initialization, as one goes from the simple GA to SAI and then MAI, the average number of iterations needed to reach the best solution generally tends to decrease. The higher computation time required by these more evolved versions is somewhat illusory, even though it is possible to conclude that the percent increase in computation time is usually higher than the percent decrease in the average number of iterations required to reach the best solution.

Table 4. Number of iterations needed to reach the best solution.

Algorithm		$n = 15$	$n = 50$	$n = 75$	$n = 100$
GA	average	268	472	486	493
	perc 50	261	482	490	496
	perc 80	396	495	496	500
GA-INI	average	26	123	151	163
	perc 50	1	9	40	105
	perc 80	1	279	343	331
GA-SAI	average	37	201	305	336
	perc 50	11	171	323	360
	perc 80	35	339	447	454
GA-SAI-INI	average	25	87	157	223
	perc 50	1	35	107	200
	perc 80	8	140	334	417
GA-MAI	average	9	105	179	228
	perc 50	4	60	131	190
	perc 80	8	180	324	401
GA-MAI-INI	average	5	69	179	184
	perc 50	1	19	132	140
	perc 80	3	99	381	322

When corresponding versions with and without initialization are compared, it can be seen that the initialization procedure allows for a substantial improvement in the average and percentile values, greatly accelerating the genetic convergence. This time it is possible to conclude that the percent increase in computation time is inferior to the percent decrease in the average number of iterations, so that the INI versions could be made to outperform their non INI counterparts, and provide equal or superior performance in solution quality by appropriately setting the *maxger* parameter.

The above results, together with the fact that the run time of the genetic algorithm with 500 generations is much longer than that of the NSearch heuristic, lead us to test the INI versions with a much lower number of generations (*maxger* was set to 50), in an effort to see if it is indeed possible to substantially reduce the computation time without incurring large penalties in solution quality. The computational results for these INI versions, both with $maxger = 50$ and with $maxger = 500$, are presented in Tables 5 and 6.

The results in Tables 5 and 6 show that setting the *maxger* parameter at 50 indeed leads to substantial savings in computation time, while avoiding large penalties in solution quality. In fact, the increase in the average percent deviation from the optimum or best heuristic value, when compared to the results with 500 generations, is quite small for the MNAI versions, though it tends to be a little higher for the non-MNAI versions. The several variants of the GA, with the exception of the GA-INI version, still outperform the NSearch heuristic, as far as solution quality is concerned, both in average percent deviation and in the number of instances for which better results are obtained.

The large gap that existed in computation times between the genetic and the NSearch procedure has been substantially reduced. With the exception of the

Table 5. Average percent deviation and computation times.

Algorithm	$n = 15$		$n = 50$		$n = 75$		$n = 100$	
	APD	ACT	APD	ACT	APD	ACT	APD	ACT
NSearch	1.2	0.0	2.6	1.8	2.1	9.9	2.4	25.5
GA-INI ($maxger = 50$)	1.4	0.2	2.5	3.4	2.5	10.7	2.7	25.1
GA-INI ($maxger = 500$)	0.8	2.0	1.9	16.7	2.2	44.0	2.5	91.8
GA-MNAI-INI ($maxger = 50$)	0.9	0.3	1.2	3.5	1.1	11.6	1.1	27.5
GA-MNAI-INI ($maxger = 500$)	0.6	2.0	1.1	16.9	1.1	44.9	1.0	94.1
GA-SAI-INI ($maxger = 50$)	0.3	0.3	1.2	4.0	1.5	12.1	1.8	27.9
GA-SAI-INI ($maxger = 500$)	0.0	2.8	0.6	26.7	1.0	74.5	1.1	148.1
GA-SAI-MNAI-INI ($maxger = 50$)	0.3	0.3	0.6	4.2	0.7	12.9	0.8	30.4
GA-SAI-MNAI-INI ($maxger = 500$)	0.0	2.8	0.4	26.9	0.5	75.1	0.5	149.9
GA-MAI-INI ($maxger = 50$)	0.0	0.4	0.5	6.7	1.0	18.1	1.0	40.5
GA-MAI-INI ($maxger = 500$)	0.0	3.9	0.2	50.8	0.4	130.5	0.5	260.2
GA-MAI-MNAI-INI ($maxger = 50$)	0.0	0.4	0.4	6.9	0.6	18.6	0.5	42.4
GA-MAI-MNAI-INI ($maxger = 500$)	0.0	3.9	0.2	50.9	0.3	130.9	0.3	261.5

Table 6. Comparison of heuristic objective function values with NSearch results.

Algorithm	$n = 15$			$n = 50$			$n = 75$			$n = 100$		
	$<$	$=$	$>$	$<$	$=$	$>$	$<$	$=$	$>$	$<$	$=$	$>$
GA-INI ($maxger = 50$)	23	49	28	18	82	0	46	0	54	43	0	57
GA-INI ($maxger = 500$)	26	52	22	51	49	0	48	0	52	47	0	53
GA-MNAI-INI ($maxger = 50$)	24	56	20	51	49	0	68	0	32	81	0	19
GA-MNAI-INI ($maxger = 500$)	27	56	17	69	31	0	71	0	29	86	0	14
GA-SAI-INI ($maxger = 50$)	32	62	6	84	16	0	74	0	26	68	0	32
GA-SAI-INI ($maxger = 500$)	37	61	2	89	11	0	83	1	16	78	0	22
GA-SAI-MNAI-INI ($maxger = 50$)	32	63	5	89	11	0	84	0	16	89	0	11
GA-SAI-MNAI-INI ($maxger = 500$)	37	61	2	91	9	0	88	1	11	90	0	10
GA-MAI-INI ($maxger = 50$)	38	61	1	91	9	0	85	0	15	86	1	13
GA-MAI-INI ($maxger = 500$)	38	62	0	95	5	0	93	0	7	95	0	5
GA-MAI-MNAI-INI ($maxger = 50$)	38	61	1	92	8	0	90	0	10	94	1	5
GA-MAI-MNAI-INI ($maxger = 500$)	38	62	0	95	5	0	93	0	7	97	0	3

versions with MAI, the genetic computation times are close to those of NSearch. It can be seen in Table 5, that the gap in computation times is reduced as the instance size increases (the GA-INI version is even faster than NSearch for the 100 job instances), which means that an increase in the number of jobs leads to higher increases in run time in the NSearch heuristic. As such, for larger instances the genetic computation time would be even closer (or indeed shorter) to that of NSearch, though this effect could possibly be offset by the need to increase the number of generations in order to maintain solution quality.

5. Conclusions

In this paper, a GA for a version of the general early/tardy problem in which no idle time may be inserted in a sequence was presented. Several versions of this algorithm, resulting from its combination with local search and initialization procedures, were developed and tested on a set of randomly generated instances, and compared to previous heuristics.

The computational results show that the genetic variants are better than the existing heuristic procedures in terms of solution quality. The results also show that significant improvements can be achieved by combining the GA with local search and initialization procedures. In fact, increasing the local search at the fitness evaluation level provides better solution values. The initialization procedure also improves the solution quality, particularly for the simpler versions of the GA. The final non-adjacent local search procedure can provide a quite visible improvement in performance for the simpler genetic versions, but it has a more limited effect on the more evolved versions. The local search at the fitness evaluation level, and particularly the initialization procedure, also greatly accelerate the convergence of the GA, therefore allowing for a reduction in the number of iterations and computation time with little effect in the solution quality.

Acknowledgment

The authors would like to thank the anonymous referees for several comments that were used to improve this paper.

References

Abdul-Razaq, TS and CN Potts (1988). Dynamic programming state-space relaxation for single-machine scheduling. *Journal of the Operational Research Society*, 39, 141–152.

Ahuja, RK, JB Orlin and A Tiwari (2000). A greedy genetic algorithm for the quadratic assignment problem. *Computers & Operations Research*, 27, 917–934.

Bailey, RN, KM Garner and MF Hobbs (1997). Using simulated annealing and genetic algorithms to solve staff scheduling problems. *Asia-Pacific Journal of Operational Research*, 14, 27–43.

Baker, KR and GD Scudder (1990), Sequencing with earliness and tardiness penalties: A review. *Operations Research*, 38, 22–57.

Bean, JC (1994), Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6, 154–160.

Goldberg, DE (1989), *Genetic Algorithms in Search Optimization and Machine Learning*. MA: Addison-Wesley, Reading.

Jaramillo, JH, J Bhadury and R Batta (2002). On the use of genetic algorithms to solve location problems. *Computers & Operations Research*, 29, 761–779.

Korman, K (1994). A Pressing Matter, Video, February, pp. 46–50.

Landis, K (1993), *Group Technology and Cellular Manufacturing in the Westvaco Los Angeles VH Department. Project Report in IOM 581, School of Business, University of Southern California.*

Lenstra, JK, AHG Rinnooy Kan and P Brucker (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1, 343–362.

Li, G (1997). Single machine earliness and tardiness scheduling. *European Journal of Operational Research*, 96, 546–558.

Liaw, CF (1999). A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem. *Computers & Operations Research*, 26, 679–693.

Ow, PS and T Morton (1989). The single machine early-tardy problem. *Management Science*, 35, 177–191.

Spears, WM and KA DeJong (1991). On the virtues of parameterized uniform crossover. In Proc. 4th Int. Conf. Genetic Algorithms, pp. 230–236.

Tan, KC, LH Lee and K Ou (2001). Hybrid genetic algorithms in solving vehicle routing problems with time window constraints. *Asia-Pacific Journal of Operational Research*, 18, 121–130.

Valente, JMS and RAFS Alves (2005a). Improved heuristics for the early/tardy scheduling problem with no idle time. *Computers & Operations Research*, 32, 557–569.

Valente, JMS and RAFS Alves (2005b). Improved lower bounds for the early/tardy scheduling problem with no idle time. *Journal of the Operational Research Society*, 56, 604–612.

Jorge M. S. Valente is Assistant Professor in the Management Department of the Faculty of Economics, University of Porto, Portugal. He holds a Ph.D degree in Management Science from the University of Porto. He has published in *Computers and Industrial Engineering, Computers and Operations Research*, *Journal of*

Manufacturing Systems and *Journal of the Operational Research Society*, among others. His current research interests include production scheduling, combinatorial optimization, heuristic techniques and agent-based computational economics.

José Fernando Gonçalves is Associate Professor in the Management Department of the Faculty of Economics, University of Porto, Portugal. He holds a PhD degree in Operations Research and Industrial Engineering from the University of Berkeley, USA. He has published in *Management Science*, *European Journal of Operational Research*, *Journal of Heuristics and Computers and Industrial Engineering*, among others. His current research interests include production scheduling, job-shop management, packing and cutting and application of genetic algorithms.

Rui A. F. S. Alves is Associate Professor in the Management Department of the Faculty of Economics, University of Porto, Portugal. He holds a PhD degree in Business Administration from the William E. Simon Graduate School of Business Administration, University of Rochester, USA. He has published in *Computers and Industrial Engineering, Computers and Operations Research, Journal of Manufacturing and Operations Management* and *Journal of the Operational Research Society*, among others. His current research interests include production scheduling, job-shop management and queueing theory.