A Hybrid Dual-Population Genetic Algorithm for the Single Machine Maximum Lateness Problem

Veronique Sels[1] and Mario Vanhoucke[1,2]

[1] Faculty of Economics and Business Administration, Ghent University, Tweekerkenstraat 2, 9000 Gent, Belgium
veronique.sels@ugent.be
[2] Operations and Technology Management Centre, Vlerick Leuven Gent Management School, Reep 1, 9000 Gent, Belgium
mario.vanhoucke@ugent.be

Abstract. We consider the problem of scheduling a number of jobs, each job having a release time, a processing time and a due date, on a single machine with the objective of minimizing the maximum lateness. We developed a hybrid dual-population genetic algorithm and compared its performance with alternative methods on a new diverse data set. Extensions from a single to a dual population by taking problem specific characteristics into account can be seen as a stimulator to add diversity in the search process, which has a positive influence on the important balance between intensification and diversification. Based on a comprehensive literature study on genetic algorithms in single machine scheduling, a fair comparison of genetic operators was made.

Keywords: Single machine scheduling, maximum lateness, genetic algorithm, dual-population structure.

1 Introduction

The single machine scheduling (SMS) problem addressed in this paper often occurs as a subproblem in solving other scheduling environments such as flow or job shops. The problem can be described as follows: there is a set N of n jobs (index j, $j = 1, 2, \ldots n$) that have to be scheduled on a single machine. The machine is assumed to be continuously available and can process at most one job at a time. The jobs may not be preempted and each job j is characterized by its processing time p_j and its due date d_j. Due to the dynamic nature of the subproblem, jobs arrive over time at the single machine, and therefore, each job is further characterized by a release time r_j. The objective is to find a schedule that minimizes the maximum lateness, L_{max}. Based on the $\alpha|\beta|\gamma$-classification scheme of [6], the problem under study can be written as $1|r_j|L_{max}$. This scheduling problem is known to be NP-hard [10]. The problem without release times, $1||L_{max}$, can be optimally solved in polynomial time, by sequencing the jobs in non-decreasing order of their due dates [9]. However, the addition of arbitrary release times makes

P. Merz and J.-K. Hao (Eds.): EvoCOP 2011, LNCS 6622, pp. 14–25, 2011.

the problem of minimizing maximum lateness on a single machine much more complex. Only a few special cases have been shown to be solvable in polynomial time. For the other cases, numerous enumerative as well as approximation approaches have been proposed in literature. A complete overview of the literature on the SMS maximum lateness problem can be found in [15]. Some noteworthy enumerative approaches are the branch-and-bound procedures of [11] and [2]. The algorithm of Carlier is considered to be one of the most efficient algorithms for the problem under study and has proven to be especially useful for solving large problem sets. The algorithm has been discussed and improved in later work by, among others, [13].

Our work is, to some extent, based on the findings of the papers mentioned above. More precisely, the conclusions of [2] and [13] concerning the difficulty of their data instances, motivated us to perform a critical analysis of the instances used in our computational experiments. The results are described in section 3.2 and further discussed in section 4.3.

The contribution of this paper is threefold. First of all, an in-depth comparison of genetic operators used in the SMS literature is done. Secondly, the effect of using a dual-population structure is investigated. Thirdly, a new data generation method is described and the obtained data instances are carefully examined and analyzed.

2 Genetic Algorithm

The genetic algorithm (GA) is a well-known search technique for solving optimization problems based on the principles of genetics and natural selection. The method was initially developed by John Holland in the 1970s [8], who was inspired by Charles Darwin's theory of evolution. Genetic algorithms (GA) have been applied to a wide variety of scheduling problems, including the single machine scheduling problem. An overview of these GA applications in the SMS literature is given in [15].

In this paper, we present a hybrid genetic algorithm for the single machine maximum lateness problem with ready times and distinct due dates. This GA is developed by means of comparing the different genetic operators described in literature. As such, the experience gained in other SMS problems is used to build an effective GA for the problem under study. Moreover, we borrowed some elements from the scatter search technique, such as the dual-population structure and the diversification generation method of [4], to enhance the important balance between diversification and intensification. As such, we not only focus on the quality of a solution, but also on the diversity of a solution. The alternative genetic operators were thoroughly tested to find the best combination for the SMS maximum lateness problem. The test results are given in section 4.2. For more information on the different genetic operators and their corresponding references in literature, we refer to [15].

In the following paragraphs, we give an overview of the hybrid dual-population genetic algorithm we implemented. A general outline is given in figure 1. In the next paragraphs, a detailed description of the different genetic operators is given.

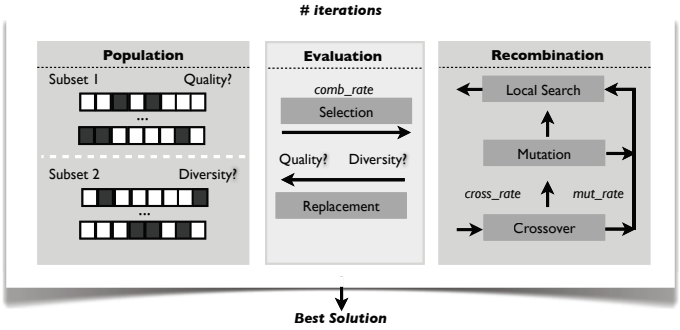

Fig. 1. Overview of GA

Solution Representation. The most natural way to represent a schedule in single machine scheduling is to use permutation encoding. In permutation encoding, every chromosome is a permutation of the n jobs in which each job appears exactly once. As such, it is ensured that each chromosome corresponds to a unique schedule and each schedule corresponds to a unique chromosome. The permutation of jobs is translated into a feasible schedule using a list engine, which simply keeps the jobs in the order of the chromosome and assigns the earliest possible starting time to them. As such, semi-active schedules are built, in which no job can be completed earlier without changing the order of processing.

Population. In order to ensure a certain degree of diversity, a dual-population structure borrowed from the scatter search framework is used in the GA. The population is split into a high quality (subset 1) and highly diverse subset (subset 2). The elements of the first subset are generated randomly and seeded with some good constructive heuristic solutions such as the Schrage heuristic [11]. The elements of the second subset are generated according to the diversification generator for permutation problems as described in [4]. The diversity in both sets is guaranteed by a distance measure calculated between every pair of solutions. This A-distance measure of [1], is equal to the sum of all absolute differences between the positions of all items in strings p and q:

$$d_{(p,q)} = \sum_i |p_i - q_i| \tag{1}$$

For the first subset, a moderate diversity threshold value has to ensure that the x unique solutions with highest fitness value are used in the initial population. The elements of the second subset only enter the initial population when a much severe diversity threshold with all of the solutions in the first subset is exceeded.

Selection. When the initial population is constructed, the algorithm has to select parent solutions that will generate new children for the next generation.

Parents can be selected from the first as well as the second subset. Solutions from the high quality subset are selected according to their fitness value. The better the fitness value, the higher the chance that the parent will be selected. Solutions from the second subset are selected based on their distance measure. Their chance of being selected increases with increasing distances. The selection methods used in the single machine scheduling literature can be generally classified in two classes. *Proportionate-based selection* selects parents according to their fitness value (distance measure) relative to the fitness value (distance measure) of the other solutions in the population. An example of such selection method is roulette wheel selection (RWS). *Ordinal-based selection* on the other hand, selects parents according to their rank within the population. Examples are tournament selection (TS) and ranking selection (RS).

Crossover. In a next step, the selected parents are recombined to create a new child or offspring. This is done by the crossover operator, which is executed with a certain probability, the *cross_rate*. Crossover can occur between solutions within the first subset or between solutions of the first and second subset. For this, we introduce a probability measure that controls the combination of high quality with highly diverse solutions, the *comb_rate*. Crossover operators for permutation encoding can be roughly classified in three classes: a class that preserves the relative order of the jobs, a class that respects the absolute position of the jobs and a class that tends to preserve the adjacency information of the jobs. As we want to minimize the maximum lateness, the relative order and/or the absolute position of each job is more relevant to the total fitness of the schedule than the adjacency information. For that reason, the crossover operators implemented in our algorithm belong to one of the first two classes. These include (linear) order crossover (OX), (uniform) order-based crossover (OBX), cycle crossover (CX), position-based crossover (PBX) and partially mapped crossover (PMX).

Mutation. After a number of generations, the chromosomes become more homogeneous and the population starts to converge. Together with the dual-population structure of our GA, the mutation operation serves as a tool to introduce diversification into a population. Mutation also occurs with a particular probability, the *mut_rate*. The mutation operators we tested include commonly used mutation operators for permutation encoding, such as a swap mutation (SM), an insertion mutation (InsM) and an inversion mutation (InvM).

Local Searches. In order to intensify our search process, we hybridize our GA with a local search algorithm. Before introducing the offspring solutions into the population, a local search technique is used to improve these solutions. A local search algorithm iteratively searches through the solution space by moving from one solution to another. It replaces the current solution by a better neighboring solution until no more improvements can be found or until some stopping criteria is met. With respect to this stopping criteria, we define a maximum number of schedules that each local search may explore during every generation of the GA. Since the risk of converging to a poor local optimum exists, the local searches are sometimes allowed to accept non-improving or neutral moves. An important

feature in the local search algorithm is the neighborhood (NH) function. This function specifies the possible neighbors of a solution. As a solution is represented by a permutation of jobs, the most common neighborhood functions used are the *insertion* and the *swap* NH [12]. An insertion NH function generates a neighbor by deleting a job from the sequence and inserting it at another position in that sequence. In a swap NH, a neighboring solution is generated by interchanging two (or more) arbitrary or adjacent jobs of the sequence. Common local search techniques from SMS literature, based on either a swap or an insertion neighborhood, include the randomized pairwise swap (RPS), the adjacent pairwise swap (APS), the 3-swap (3S), the insertion (INS) and the largest cost insertion (LCI).

Replacement and Termination Condition. Once the new offspring solutions are created, they have to be inserted in the population. We use an incremental replacement strategy, where the offspring solutions replace the least fit solutions of the current population. If the solutions are not allowed into the new population, they are checked on their diversity using equation 1. If their distance is greater than the smallest distance in the current diverse subset, the solution is accepted in the new population. The algorithm is stopped within the time limit of one second or when the optimal solution is found. A solution is optimal when the job with the largest lateness starts at its release time or when the solution equals the lower bound.

3 Data Generation

3.1 Generation of Instances

In the literature of the $1|r_j|L_{max}$ problem, no standard data set can be found. This motivated us to analyze the methods described in the literature and translate them into two simple generation methods. A distinction was made based on how the due dates of the jobs are generated. The first method generates due dates that depend on the release and/or processing times of the jobs. The method is based on the due date assignment methods described in the paper of [5]. These *dependent* due dates are uniformly distributed between $r_j + kp_j$ and $r_j + kp_j + q$. The parameter k represents the due date tightness, while parameter q defines the slack allowance. The larger k and q, the more slack a job has to be scheduled between its release time and its due date. The second generation method generates due dates that do not depend on the jobs' processing times and/or ready times, but only on the sum of the processing times of all jobs. In general, the *independent* generation method is based on the widely used techniques of [14] and [7]. These independent due dates were generated from the uniform distribution $U[a\sum p_j, b\sum p_j]$, where a and b are (wide-ranging) parameters that define the range and location of due dates relative to the period that the machine processes the jobs ($a \leq b$).

These two generation methods result in two different data sets, set I and set II, for dependent and independent due dates respectively. The problem size is

Table 1. Instance parameters

Parameter	Set I	Set II
processing time p_j	U[1,100]	U[1,100]
release time r_j	U[0, $l\sum p_j$]	U[0, $l\sum p_j$]
	l = 0, 0.25, 0.5, ..., 4	l = 0, 0.25, 0.5, ..., 2
due date d_j	U[$r_j + kp_j$, $r_j + kp_j + q$]	U[$a\sum p_j$, $b\sum p_j$]
	k = -1, 0, 5, 10	a = 0, 0.25, ..., 1
	q = 0, n, 5*n, 10*n	b = 0, 0.25, ..., 1.5

set to 100 jobs. The job processing times p_j are uniformly distributed integers between 1 and 100. The release times r_j are integers generated from the uniform distribution $U[0, l\sum p_j]$ where l defines the range of distribution of r_j. The smaller l, the more frequent a job arrives in the system. An overview of the parameter settings can be found in table 1. In both sets, every combination of release and due date parameter values leads to a certain problem class. This results in 272 problem classes for Set I and 225 classes for Set II. For each problem class, ten instances were generated, resulting in a total of 4,970 instances[1].

3.2 Data Analysis

An analysis of the data obtained is performed to make a distinction among the difficulty levels of the instances and thus, to provide more insights in the performance of the genetic algorithm. Figure 2 illustrates this analysis. On the left axes, the due date ranges are given by the different parameter values (i.e. k and q for set I and a and b for set II). The release times ranges, with parameter l, are shown on the bottom axes. The cells in the figures represent the different problem classes, each cell containing ten problem instances. The shaded areas designate the different difficulty levels of the corresponding instances. The darker the shade, the less difficult the class of instances is. These difficulty levels are based on computational experiences performed by analyzing the solution obtained by the Schrage heuristic [11] for every instance. From this analysis, it could be observed that instances of some problem classes were already solved optimally. As a consequence, we could presume that some instances were rather 'easy' and others were rather 'hard' to solve. We conjectured that the more instances per problem class could be solved with the Schrage heuristic, the less difficult that class of instances was.

As a result, the difficulty of the problem classes can be subdivided in different levels. The first level contains problem classes in which every instance was solved optimally by the Schrage heuristic. The *theoretical optimal* instances, denoted by TO, have release times equal to zero ($l = 0$), due dates equal to their release times ($k = q = 0$) or common due dates ($a = b$), and are known to be solved optimally by the Schrage heuristic. The *empirical optimal* instances were observed to be solved optimally by the EDD-rule, but no theoretical funded explanation can be given. We presume that their optimality results from the fact that their

[1] The problem instances can be downloaded from the website *www.projectmanagement.ugent.be/machinescheduling.html*.

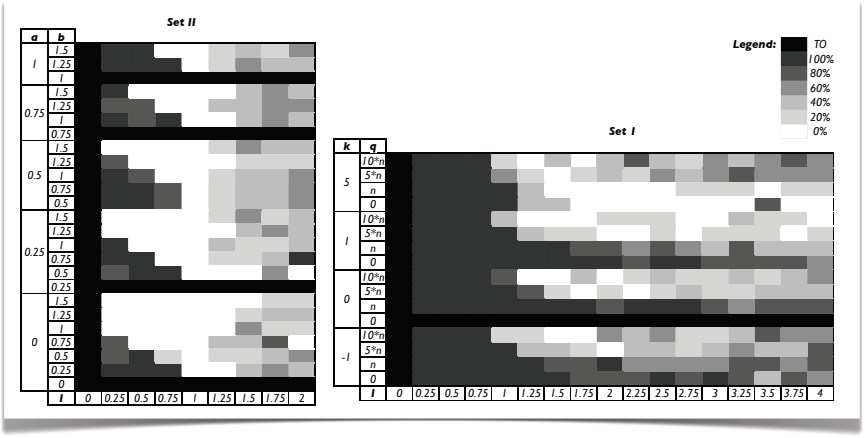

Fig. 2. Data analysis for both data sets based on the due date and release time parameter values

release time and/or due date ranges are closely related to those of the theoretical optimal instances. These classes are represented by the 100% regions in figure 2, denoting that all instances were solved optimally. The next level of difficulty include problem classes with instances for which the Schrage heuristic mostly was able to find the optimal solution. These are called the *empirical easy* instances. For 60% to 90% of the instances, the optimal solution was found. The third level are the *empirical moderate* problem classes. Some instances, 20% to 50%, were solved optimally, but in general, an optimal solution could not be found. The last level are the *empirical hard* problem instances, for which the Schrage heuristic was not able to find the optimal solution. At first sight, when looking at the shaded areas, it seems that the instances of set I are less difficult than the instances of set II. This conjecture will be assessed in a computational experiment described in section 4.3.

4 Computational Experiments

4.1 Parameter Fine-Tuning

In order to refine our hybrid dual-population genetic algorithm, decisions with respect to the parameter values have to be made. Examples are the population size, the combination, crossover and mutation rates. These parameters were fine-tined by performing tests in cycles. Each cycle, a single parameter was chosen to be fine-tuned, while the other parameters were set to a certain value. All possible values for that parameter were tested and its best value was fixed before going to the next parameter. This process is repeated for all parameters until no more improvement was found.

In table 2, the different test values of all parameters are given, together with their best value for both data sets. The table shows that, for both sets, the

Table 2. Parameter values used in GA

Parameter	Test values	Best values Set I	Best values Set II
pop_size	n, $2n$, $5n$, $n/2$, $n/5$	$n/5$	$n/2$
comb_rate	0, 0.1, ..., 1	0.85	0.85
cross_rate	0, 0.1, ..., 1	0.5	0.6
mut_rate	0, 0.1, ..., 1	0.1	0.3

population size is relatively small thanks to the intelligence of the algorithm and the dual-population structure. The combination rate, which determines the probability of combining a high quality with a high diverse population element, is set to 85%. This means that there is 85% chance on combining elements within the first subset of the population and 15% chance on combining elements from the first with the second subset of the population. The optimal crossover and mutation rates are equal to 50% and 60%, and 10% and 30%, respectively.

4.2 Operator Selection

In this section, the search for the best combination of genetic operators is discussed, given the ideal parameters of the previous section. We performed a full factorial design where every possible combination was tested in cycles. For each genetic operator, the best alternative was obtained by fixing the other operators to their best alternative. In table 3, the results of the design are given. The values in the table are obtained by calculating the relative performance of the genetic algorithm. The relative performance (RP) of the genetic algorithm was measured by the deviation of the objective values found by the GA with a lower bound and is equal to

$$RP = \frac{\sum_{GA} - \sum_{LB}}{\sum_{LB}}, \tag{2}$$

where $\sum_{GA}$ is the sum of the objective values over some set of instances obtained by the GA and $\sum_{LB}$ is the sum of the lower bounds over the same set of instances. This lower bound was obtained by combining four lower bound calculations from literature as described in [11], [2] and [3].

For set I, the best combination is to use the *tournament* selection method together with the *position-based* crossover, the *single swap* mutation and the *largest cost insertion* local search algorithm. The combination of the *tournament* selection method with the *cycle* crossover, the *inversion* mutation and the *randomized pairwise swap* improvement heuristic turned out to be the best for set II. However, when looking at the table, it can be noticed that in comparison with set I, the differences between the operators of set II are relatively small. The choice of genetic operators seems to be of little importance for set II (i.e. with independent due dates). Moreover, the deviations from the lower bound in general are very small. This makes us presume that the instances of set II have a lower difficulty level than figure 2 of section 3.2 would imply. In the next section, the difficulty of these instances will be therefore further examined.

Table 3. Operator selection - Relative performance (%)

Operator		**Set I**	**Set II**
Selection:	TS	**0.1348**	**0.0058**
	RWS	0.1493	0.0073
	RS	0.2966	0.0073
Crossover:	PMX	0.1952	0.0073
	OX	0.2010	0.0073
	PBX	**0.1348**	0.0073
	OBX	0.1362	0.0073
	CX	0.2787	**0.0058**
Mutation:	SM	**0.1348**	0.0073
	InsM	0.1488	0.0073
	InvM	0.1411	**0.0058**
Local search:	RPS	0.1884	**0.0058**
	APS	0.1493	0.0073
	INS	1.4150	0.2948
	3S	0.4995	0.0073
	LCI	**0.1348**	0.0073

4.3 Computational Results

In this section, we compare the performance of our algorithm, with the performance of the Schrage heuristic with (*Schrage_LS*) and without local search (*Schrage*) and the Multi-Start algorithm (*MultiS*) in order to obtain benchmark results[2]. The Multi-Start algorithm starts with (1,000) randomly chosen solutions followed by a hill-climbing technique, which seeks to improve each initial schedule to its local optimum. Three versions of our algorithm were tested, the genetic algorithm with a single-population structure (only high quality solutions) (*GA*), the dual-population GA (*2PGA*) and the hybrid dual-population GA with the best performing local search (*2PGA_LS*).

The computational results are listed in table 4, which summarizes the relative performances of all heuristics. We make a distinction according to the difficulty levels described before. In doing so, certain effects become more clear when only 'hard' instances (e.g. the -20% instances are assumed to be the hardest problem instances) are considered. The results in both tables reveal the contribution of the various solution approaches, the local search procedures and the various operators embedded in the genetic algorithm. They can be summarized along the following lines:

- The contribution of an intensive multi-pass search versus a quick single-pass search leads to obvious conclusions. The single-pass Schrage algorithm performs worst compared to the more time-consuming multi-pass algorithms (GA, 2PGA, 2PGA_LS and MultiS). However, the results show that the efficient Schrage_LS algorithm is better than the time-consuming MultiS algorithm for the sets with empirical and/or theoretical optimal instances (*Full set* and *-TO* columns). These sets contain instances for which the Schrage algorithm is proven to generate optimal solutions, which could often

[2] The algorithms described were coded in Visual Studio C++ and run on a 2.6 GHz Intel Pentium Processor.

not be found by the intensive MultiS search. It is for this very reason that the size of the data sets is further decreased to only the more difficult instances. In doing so, we avoid that many instances from the sets can be quickly solved by single-pass heuristics or straightforward extensions of these heuristics.
- The contribution of local search algorithms is clearly shown in the table, since all solution procedures with LS always outperform their non-LS version: the hybrid 2PGA_LS outperforms 2PGA, Schrage_LS outperforms Schrage and MultiS (which also contains the LS procedures) outperforms both Schrage and 2PGA without local search.
- The contribution of the dual-population structure can be seen by comparing the performance of the GA with the 2PGA. These results show, that the incorporation of a dual population structure is a crucial tool that allows the GA the possibility to explore the solution space more efficiently.
- The contribution of the genetic algorithm operators can be best shown when comparing the 2PGA_LS with the MultiS algorithm. The 2PGA_LS outperforms the MultiS performance, even when the 2PGA and 2PGA_LS algorithms have been truncated after 1 second, while the MultiS search algorithm needed on average 1 minute to evaluate the 1,000 random solutions. Moreover, the hybrid dual-population genetic algorithm (2PGA_LS) always outperforms all other solution procedures, both for the dependent (set I) and independent (set II) data sets.

It should be noted that the differences between the relative performances of table 4 are less clear for the data of set II compared to set I. This is in line with the findings of the previous section. This could possible lead to the conjecture that most instances with independent due dates of set II are relatively easy to solve, which supports the findings of [2] and [13]. In the paper of [2], 999 out of the 1,000 instances described were solved optimally. Seemingly an outstanding result, but Carlier noted that "in nine cases out of ten, either the Schrage solution or the schedule when the critical job was scheduled after the critical set J was optimal". Moreover, he tested instances up to 10,000 jobs, but even those instances were solved optimally with only one node in the tree. This was

Table 4. Comparison of methods - Relative performance (%)

Data set	Method	Full set	- TO	-100%	-80%	-60%	-40%	-20%
Set I	GA	1.4895	2.4968	25.4976	46.7115	65.1439	133.7975	339.6232
	2PGA	0.9763	1.2515	12.1321	25.8566	47.5332	99.4174	224.8983
	2PGA_LS	0.2538	0.4255	4.3452	7.8404	11.0952	20.3734	46.3563
	Schrage	1.6620	2.7859	28.4500	52.1908	73.0150	149.0884	376.5309
	Schrage_LS	0.6282	1.0531	10.7543	19.6902	27.9287	55.5257	147.4370
	MultiS	1.0771	1.7020	8.3532	13.5248	19.2784	31.0752	76.5863
	#inst	2720	2400	1650	1320	1070	730	360
Set II	GA	0.1391	0.1836	0.1882	0.1832	0.2159	0.3174	0.5241
	2PGA	0.0951	0.1186	0.1323	0.1544	0.1795	0.2004	0.3941
	2PGA_LS	0.0272	0.0358	0.0368	0.0282	0.0322	0.0458	0.0703
	Schrage	0.4915	0.6485	0.6649	0.6768	0.7616	1.0123	1.4161
	Schrage_LS	0.0488	0.0643	0.0660	0.0565	0.0651	0.0879	0.1225
	MultiS	3.8533	0.4263	0.1779	0.0701	0.0828	0.1257	0.2139
	#inst	2250	1600	1400	1290	1140	860	630

conformed by the study of [13], who stated that "Carlier reported the remarkable performance of its algorithm on a test set of 1,000 instances, but he also pointed out that the large majority of those instances was easy". They extended the set of instances, with the same parameters, to 500,000 and they concluded that with Carlier's algorithm, only 101 instances remained unsolved. This also indicates that the instances studied are not that challenging. To confirm these conjectures, a further thorough analysis of the instances of both data sets was done. Through empirical testing, we have noticed that the instances of set II often had the characteristic that the lateness of the job j that was responsible for the L_{max} was equal to the difference between d_j and $r_j + p_j$. As a consequence, in order to obtain an optimal schedule, this job has to be fixed on its release time, while the order of the other jobs is less important. Hence, many permutations will lead to the optimal solution, as long as this single job starts at his release time. This makes the problem instances somewhat easier to solve, because any greedy sequence of simple moves (swap and/or insertion) performed on any non-optimal schedule can easily improve the schedule quality. Comparing this with the findings of section 3.2, there seems to be no direct link between the Schrage heuristic performance and the actual difficulty of the instances of the independent data set, as the previous section presumed. Because of the independency of the due dates, there is no relation between release times, processing times or due dates. This makes solving these instances harder for a single pass heuristic (i.e. the Schrage heuristic) that is more or less based on these relationships. This explains the rather poor performance of the heuristic on the relative 'easy' instances of set II.

5 Conclusions

In this paper we have examined the single machine maximum lateness problem with distinct release times and due dates. A hybrid dual-population genetic algorithm was developed by means of comparing different genetic operators from the SMS literature. Various alternatives for the operators were tested in a full factorial design to find the best combination for the problem under study. The computational experiments were performed on two diverse data sets, with dependent or independent due dates, which are a summary of the methods described in literature. The instances in both sets were analyzed to examine their difficulty levels. The results indicated that the instances with an independent due date were not that challenging. This was justified by literature and our own experiments. Comparison was made with the Schrage heuristic and a Multi-Start algorithm. The test results illustrate the contribution of the various solution approaches, the local search procedures, the various operators and the dual-population structure embedded in the genetic algorithm. Possible directions for future research include employing other metaheuristics and extending the problem with setup considerations and batching. Moreover, the inclusion of the problem in solving more complex environments such as job shop or flow shop scheduling is an interesting field of study.

References

1. Campos, V., Laguna, M., Marti, R.: Context-independent scatter and tabu search for permutation problems. Journal on Computing 17, 111–122 (2005)
2. Carlier, J.: The one-machine sequencing problem. European Journal of Operational Research 11, 42–47 (1982)
3. Chang, P.-C., Su, L.-H.: Scheduling n jobs on one machine to minimize the maximum lateness with a minimum number of tardy jobs. Computers & Industrial Engineering 40, 349–360 (2001)
4. Glover, F.: A template for scatter search and path relinking. In: Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (eds.) AE 1997. LNCS, vol. 1363, pp. 13–54. Springer, Heidelberg (1998)
5. Gordon, V.S., Proth, J.-M., Chu, C.: Due date assignment and scheduling: SLK, TWK and other due date assignment models. Production Planning & Control 13(2), 117–132 (2002)
6. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy-Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5, 287–326 (1979)
7. Hariri, A.M.A., Potts, C.N.: Single machine scheduling with batch set-up times to minimize maximum lateness. Annals of Operations Research 70, 75–92 (1997)
8. Holland, J.: Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor (1975)
9. Jackson, J.R.: Scheduling a production line to minimize maximum tardiness. Management Science Research Report 43, University of California, Los Angeles (1955)
10. Lenstra, J.K., Rinnooy-Kan, A.H.G., Brucker, P.: Complexity of machine scheduling problems. Annals of Discrete Mathematics 1, 343–362 (1977)
11. McMahon, G., Florian, M.: On scheduling with ready times and due dates to minimize maximum lateness. Operations Research 23, 475–482 (1975)
12. Michiels, W., Aarts, E., Korst, J.: Theoretical Aspects of Local Search Series. In: Monographs in Theoretical Computer Science, An EATCS Series (2007)
13. Pan, Y., Shi, L.: Branch-and-bound algorithms for solving hard instances of the one-machine sequencing problem. European Journal of Operational Research 168, 1030–1039 (2006)
14. Potts, C.N., Van Wassenhove, L.N.: A branch and bound algorithm for the total weighted tardiness problem. Operations Research 33(2), 363–377 (1985)
15. Sels, V., Vanhoucke, M.: A genetic algorithm for the single machine maximum lateness problem. Technical report, Ghent University (2009)