# A comparative study of solution representations for the unrelated machines environment

Ivan Vlašić[a], Marko Đurasević[a,*], Domagoj Jakobović[a]

*ivan.vlasic2@fer.hr, marko.durasevic@fer.hr, domagoj.jakobovic@fer.hr*

[a] *University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia*

**Abstract**

Scheduling problems are quite difficult to solve since in many cases no exact algorithms exist which can obtain the optimal solution in a reasonable amount of time. Therefore, these problems are often solved by using various metaheuristic methods, like genetic algorithms. To use these methods, the first step which needs to be performed is to define an encoding scheme that will be used to represent the solutions. Until now, several encoding schemes were proposed for the unrelated machines environment, each of which comes with its own benefits and drawbacks. However, the performance of metaheuristic methods depends on the applied encoding scheme. Unfortunately, no extensive research was performed in the literature to compare different solution representations for the unrelated machines scheduling problem. Therefore, the choice of the solution representation used is mostly provisional and is usually not based on any existing knowledge of how it would perform on the considered problem. This can cause the algorithms to obtain suboptimal results, which can lead to wrong conclusions about the performance. Thus, the goal of this paper is to test seven solution representations that were used in previous studies to represent solutions for the unrelated machines scheduling problem. The selected solution representations were tested for optimising four scheduling criteria, while additionally measuring the execution time of the genetic algorithm when using each of the encodings. The obtained results demonstrate that the encoding which is based on the permutation of jobs obtains the best results, making it the superior encoding scheme for this type of scheduling problem.

*Keywords:* Unrelated machines environment, genetic algorithms, solution representations, scheduling

## 1. Introduction

Scheduling is a process by which a certain number of jobs are allocated to a set of machines, in a way that one or more user-defined criteria are opti-

*Corresponding author

mised (Pinedo, 2012). In the unrelated machines environment, each job needs to be allocated to only one of the available machines. However, each job has a different execution time for each of the available machines, meaning that the choice of the machine on which a job will be executed can have a high impact on the generated schedule. Scheduling in the unrelated machines environment can be found in many practical real-world examples, such as scheduling jobs on multiprocessor computers, airplanes on landing lanes in airports, operations to operating rooms in hospitals, jobs in circuit board and semiconductor manufacturing, and many other (Fanjul-Peyro & Ruiz, 2012; Lee et al., 2013; Wang et al., 2013). Although scheduling problems can be solved by using various optimisation methods, heuristic and metaheuristic methods are most often used to obtain solutions. The reason is that most scheduling problems are NP-hard, therefore an algorithm that could solve such problems optimally in a reasonable amount of time is unknown. However, metaheuristic algorithms can obtain solutions of good quality in a relatively short amount of time, which makes them suitable for solving various scheduling problems.

Although a variety of different metaheuristic methods were proposed in the literature, the genetic algorithm (GA) (Goldberg, 1989; Mitchell, 1998; Eiben & Smith, 2015) represents one of the most popular and widely used metaheuristic algorithms. To be able to apply GAs for solving a certain scheduling problem, it is mandatory to define how the solutions to this problem are represented in the algorithm. The choice of the solution representation has a large impact not only on the effectiveness, execution time, and memory consumption of the algorithm but also on the genetic operators which can be used for adapting the solutions. Therefore, it is important to select the appropriate solution representation for solving the problem at hand. Although many different solution representations were proposed for solving scheduling problems in the unrelated machines environment until now no exhaustive study was conducted which compares the different solution representations and outlines the benefits and drawbacks of each. Thus, when selecting the solution representation which will be used, researchers are mostly left to select the representation based on their intuition and prior experience. Therefore, in many papers the solution representation is selected without any justification or comparison to alternative representations. This can result in the selection of an inappropriate solution representation for the considered problem, which can consequentially lead the algorithm to obtain poor results. This can have significant influence on the obtained conclusions, since they might not be valid if another representation would have been used. As a result, it would be possible that some studies come to the wrong conclusions about the effectiveness of an algorithm, just because an inappropriate representation was used. Therefore, the choice of the right representation for the considered problem is of great importance to ensure that the results and conclusions that are obtained in studies are relevant.

The objective of this paper is to collect several solution representations that were previously used for solving scheduling problems in the unrelated machines environment and to perform an evaluation of these representations. The reason why the results from existing studies could not simply be used for comparing

the solution representations lies in the fact that those studies used different problem instances and optimised various scheduling criteria. Therefore, it was required to perform the tests of all the representations on the same problem instances and under the same conditions. To obtain a better and more objective conclusion about the performance of each solution encoding, they were used to optimise four scheduling criteria. Furthermore, the execution time of the GA for each of the tested encodings was also analysed, since it can have an impact on the choice of the appropriate solution representation. Based on the obtained results for the optimised criteria and the execution time for the GA, the paper outlines the benefits and drawbacks of each encoding and draws certain conclusions about each of them. Therefore, depending on the situation and requirements, it should be possible to select the representation which fulfils the requirements and provides the best possible results. The analyses and conclusions presented in this paper should give other researchers a good starting point for selecting the most appropriate solution representation. This should improve the research in the area of unrelated machines environment, since it would reduce the number of studies which would obtain results and conclusions which might be less informative and incomparable, simply because an inappropriate representation was selected.

The rest of the paper is structured as follows. Section 2 provides an overview of the existing research focused on solving the unrelated machines scheduling problem, especially when using various metaheuristic methods. A short description of the unrelated machines scheduling problem is given in Section 3. The solution representations tested in the paper are described in Section 4. Section 5 describes the design of the experiments used for evaluation, and provides details on the parameter values which were used by each of the solution representations. The results obtained by each of the tested solution representations are presented and analysed in Section 6. Section 7 provides a discussion on the benefits and drawbacks of the tested solution representations. Finally, Section 8 gives a brief conclusion and outlines the possibilities for future work on this and similar topics.

## 2. Literature overview

Solving various kinds of scheduling problems has until now been an extensively researched field (Allahverdi et al., 1999, 2008; Hart et al., 2005; Branke et al., 2016). Because most scheduling problems are NP-hard (Pinedo, 2012), exact methods cannot be used for larger problem instances. On the other hand, approximation methods are hard to design and cannot be defined for all problems. Therefore, the research in scheduling has largely focused on developing new heuristic methods for solving various scheduling problems. Unfortunately, such heuristics are difficult to design and are usually specialised for solving only a specific type of scheduling problem. As a result, a lot of research focused on applying different metaheuristic methods for solving scheduling problems. The benefit of using metaheuristics is that they can easily be adapted for solving

various kinds of scheduling problems. Although these algorithms do not guarantee that they will obtain the optimal solution for the problem, they mostly obtain solutions of good quality in a reasonable amount of time. The unrelated machines environment did not receive the same amount of attention as some other machine environments, although it appears in various real-world situations. Nevertheless, a wide range of methods were proposed for solving the unrelated machines scheduling problem, ranging from exact methods (Graham et al., 1979; Rocha et al., 2008; Pinedo, 2012; Wotzlaw, 2012), approximation methods (Graham et al., 1979; Lenstra et al., 1990; Chen et al., 1998; Wotzlaw, 2012), problem-specific heuristics (Fanjul-Peyro & Ruiz, 2010, 2011; Cota et al., 2014; de C. M. Nogueira et al., 2014), manually designed dispatching rules (Morton & Pentico, 1993; Maheswaran et al., 1999; Braun et al., 2001; Pinedo, 2012; Đurasević & Jakobović, 2018), and automatically designed dispatching rules (Đurasević et al., 2016; Đurasević & Jakobović, 2017a,b).

Glass et al. (1994) consider the problem of scheduling jobs in the unrelated machines environment when optimising the makespan criterion. The authors apply tabu search, simulated annealing, and a GA to solve the considered problem. Srivastava (1998) develops a tabu search heuristic for minimising the makespan criterion in the unrelated machines environment. Kim et al. (2002) apply the simulated annealing algorithm for solving problems in the unrelated machines environment with setup times. The authors have proposed six methods for constructing the neighbourhood of the current solution, and show that these methods improve the performance of the simulated annealing method. The simulated annealing method was further used by Kim et al. (2003) to minimise the total weighted tardiness criterion in batch scheduling of unrelated machines environment. Kim & Shin (2003) propose a restricted tabu search method for minimising the maximum lateness of jobs in the unrelated machines environment with sequence-dependent setup times, release times, and due dates. The problem of scheduling printed circuit boards on unrelated parallel machines was considered by Hop & Nagarur (2004). The authors minimised the total makespan objective by using the proposed composite GA. Logendran et al. (2007) apply the tabu search algorithm for the unrelated machines scheduling problem with sequence-dependent setup times. In their paper, the authors have developed four different initial solution construction mechanisms and measure their influence on the quality of the final solution. Vallada & Ruiz (2011) propose a GA for the unrelated machines environment with sequence-dependent setup times for optimising the makespan criterion. In their work, the authors use the machine list encoding to represent the solutions and incorporate a local search operator to improve the performance of the GA.

Raja et al. (2008) focused on solving the unrelated parallel machines environment in which they minimised the total earliness and total tardiness criteria. For solving the aforementioned problem the authors propose a new technique, which is a combination of the GA and fuzzy logic. The experimental results demonstrated the effectiveness of this technique when compared to other methods. Behnamian et al. (2009) propose a hybrid algorithm for optimising the parallel unrelated machines problem with sequence-dependent setup times. The

hybrid algorithm combines the ant colony optimisation method with simulated annealing and variable neighbourhood search methods. Chyu & Chang (2010) propose a competitive evolution strategy memetic algorithm for optimising the total weighted tardiness and flowtime criteria in the unrelated machines environment. Balin (2011) used a GA to optimise the makespan criterion in the unrelated machines environment. In this study, the matrix encoding is used to represent solutions of the scheduling problem. The algorithm was also enhanced by a new crossover operator, and it performed better than some standard scheduling methods. Chang & Chen (2011) deal with an unrelated machines scheduling problem with setup times and the objective of minimising the makespan. The authors develop a set of dominance properties that enables the algorithm to obtain near-optimal solutions. By combining a GA with these dominance properties the authors define a new metaheuristic that obtains optimal solutions for smaller problem instances and outperforms other algorithms for larger ones.

Haddad et al. (2012) consider the minimisation of the makespan criterion in the unrelated machines environment with setup times. The authors propose a new method called GARP, which is based on the combination of a GA with the variable neighbourhood descent and path relinking methods. In this work, the authors use the machine list encoding to represent the solutions. Costa et al. (2013) address the problem of scheduling in the unrelated machines environment with sequence-dependent setup times and limited human resources. In this work, the authors apply the permutation encoding, but also propose a multi-encoding scheme in which they use several permutation encodings to denote the solution. Lee et al. (2013) apply a tabu search algorithm for minimising the total tardiness criterion in the unrelated machines environment with setup times. The authors used the machine list encoding to represent the solutions and showed that the applied tabu search method achieved a better performance than other tested methods. Lin et al. (2013) applied the ant colony optimisation algorithm for minimising the total weighted tardiness. The authors introduce several new ideas by which they enhance the performance of the algorithm. de C. M. Nogueira et al. (2014) propose a hybrid GRASP heuristic and apply it to solve the unrelated machines scheduling problem in which they minimise the earliness and tardiness criteria. The experiments show that the proposed hybrid GRASP heuristic performs well when compared to other methods. Đurasević & Jakobović (2016) apply a GA with two solution representations for solving four scheduling criteria in the unrelated machines environment, and compare the results and execution times of the tested GA with those of several dispatching rules.

## 3. Problem formulation

In the unrelated machines environment the goal is to schedule each of the $n$ available jobs on one of the $m$ machines. To denote a specific job, the index $j$ will be used, while the index $i$ will be used to denote a specific machine. In the basic definition of the problem, each machine can execute only one job at a time. Once a job is scheduled on a machine it cannot be rescheduled and must

be executed until the end on the allocated machine. The particularity of this environment is that each job has a different execution time defined for each of the available machines. The values for these processing times have no apparent relation to either jobs or machines, thus making the problem challenging to solve. The processing times are denoted by $p_{ij}$, where $i$ denotes a machine in the system, and $j$ denotes a concrete job. Jobs are rarely present from the start of the execution of the system but are rather expected to arrive at a certain point in time into the system. This moment in time is called the release time of the job and is denoted as $r_j$. Besides, jobs usually need to be finished until a certain point in time, which is called the due date of a job and is denoted as $d_j$. Although a job is allowed to finish after its due date, by doing so it invokes a certain penalty which needs to be minimised as much as possible. Finally, depending on the criteria which are optimised, it is also possible to define a weight for each job, denoted as $w_j$, which specifies the importance of each job.

The previously outlined job properties denote the input to a certain algorithm when creating a schedule. After the schedule is constructed, a set of output properties that are calculated based on the constructed schedule, can be defined for each job. These output properties are used to specify different scheduling criteria. The most basic output property of each job is the time when the job finishes with its execution, which is called the *completion time* and is denoted as $C_j$. Based on that property it is possible to calculate the *flowtime* of each job, which represents the amount of time that each job spent in the system, and is calculated as $F_j = C_j - r_j$. For each job it is also possible to calculate its *tardiness* as $T_j = \max(0, C_j - d_j)$, which defines the amount of time that the job spent executing after its due date. Finally, the *unit penalty*, which denotes whether the job is tardy or not, is defined for each job as $U_j = \begin{cases} 1 : T_j > 0 \\ 0 : T_j = 0 \end{cases}$.

Based on the previously outlined output properties, various scheduling criteria can be defined (Allahverdi et al., 1999; Leung, 2004; Allahverdi et al., 2008; Pinedo, 2012; Baker & Trietsch, 2013). One of the most commonly optimised criteria is the *makespan* $C_{max} = \max_j C_j$, which is defined as the largest completion time of all jobs in the system. The *total flowtime* is defined as the sum of all job flowtimes in the system: $Ft = \sum_j F_j$. The *total weighted tardiness* is defined as the sum of the tardiness value of each job multiplied by the weight of each job: $Twt = \sum_j w_j T_j$. Finally, the *number of tardy jobs* is defined as the sum of weights of all tardy jobs in the system: $Nwt = \sum_j w_j U_j$.

All the scheduling problems considered in this paper are executed under static conditions. This means that all the information about the scheduling problems are known in advance. The GA is then executed on these problems to find the best possible schedule, based on which jobs will be allocated on the available machines. Therefore, the schedule is constructed in advance, before the system even starts with its execution.

Table 1: Properties of the problem instance used for analysis

| Job index $j$ | $r_j$ | $d_j$ | $w_j$ | $p_{0j}$ | $p_{1j}$ | $p_{2j}$ |
|---|---|---|---|---|---|---|
| 0 | 15 | 30 | 0.9 | 10 | 7 | 7 |
| 1 | 10 | 23 | 0.07 | 9 | 9 | 8 |
| 2 | 1 | 13 | 0.87 | 11 | 13 | 12 |
| 3 | 20 | 37 | 0.06 | 9 | 6 | 11 |
| 4 | 22 | 35 | 0.06 | 10 | 14 | 11 |
| 5 | 5 | 25 | 0.26 | 11 | 15 | 16 |
| 6 | 7 | 20 | 0.78 | 15 | 12 | 9 |

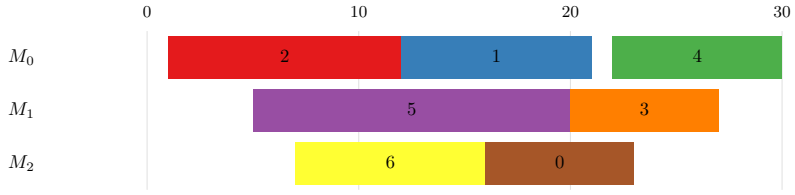



Figure 1: Example of a schedule for the previous scheduling problem

## 4. The GA and solution representations

To test different solution representations, a simple GA will be used. At the start of the algorithm all the individuals are generated randomly, however, in the individual initialisation process it is always ensured that all individuals represent feasible solutions. In the evolution process, the GA behaves as a steady-state GA, with the tournament selection. The tournament size of three individuals was used, which means that three individuals from the population are randomly selected into the tournament. Out of these three individuals, the crossover operator will be performed on the best two to create a new individual which will be additionally mutated with a certain probability and will then replace the worst individual in the tournament. This entire procedure will be performed until a certain number of function evaluations are reached.

The solution representations which will be used by the GA are permutation encoding, permutation encoding with machine list, permutation encoding with machine count, random key encoding, floating point encoding, matrix encoding, and machine list encoding. To better denote the differences between all the representations, the solution for a small scheduling problem will be represented by using each of the tested solution representations. Table 1 represents the properties of the problem instance, while Figure 1 represents a solution to this scheduling problem. The problem consists out of seven jobs which need to be scheduled on one of the three available machines. The rest of this section describes each of the individual solution representations.

### *4.1. Permutation encoding*

The *permutation encoding* (PE) is one of the most simple encodings which are used (Costa et al., 2013). In this encoding, the entire solution is represented by using only a single list consisting out of integer numbers. The size of this list is equal to the number of jobs $n$, and each number in this list represents the index $j$ of a concrete job. This list of indices represents the sequence in which jobs will be scheduled, meaning that the jobs at the start of the list will be scheduled first, while the jobs at the end of the list will be scheduled last. However, the information in this list is by itself not enough to construct the complete schedule, since it does not specify the machine on which each job needs to be scheduled. Therefore, it is essential to define a procedure by which the machine on which a certain job should be scheduled can be determined. The procedure that will be used in this paper will schedule the current job on the machine on which the job would complete with its execution the soonest.

Figure 2 shows the solution represented by using this encoding for the example in Table 1. The example shows that job $J_6$ needs to be scheduled first. However, it is still required to determine the machine on which the job will be executed. Since at the release time of the job all the machines are free, the job is scheduled on the machine on which it has the smallest processing time, which would, in this case, be machine $M_2$. The next job in the list is job $J_2$. To decide on which machine this job should be scheduled, the completion times of the job on each of the machines are calculated. Since no jobs are scheduled on machines $M_0$ and $M_1$, the completion time on those machines is calculated as $r_j + p_{ij}$. Therefore, the completion time of job $J_2$ on machine $M_0$ would be 12, while on machine $M_1$ it would be 14. On the other hand, since at least one job is already scheduled on machine $M_2$, it is first required to determine the time at which all jobs finish with their execution on that machine (denoted as $mr$), which would, in this case, be 13. The completion time of the job on that machine is calculated as $\max(mr, r_j) + p_{ij}$, which would amount to 25. Since the job would complete the fastest on machine $M_0$, it is scheduled on that machine. The remaining jobs in the list are scheduled in the same way, which would result in the schedule presented in Figure 1.

| 6 | 2 | 5 | 1 | 0 | 4 | 3 |
|---|---|---|---|---|---|---|

Figure 2: Schedule represented by PE

### *4.2. Permutation encoding with machine list*

The permutation encoding with machine list (PEML) is a simple extension of PE, which also enables the association of each job to a certain machine (Đurasević & Jakobović, 2016). In this representation, in addition to the list representing the permutation of the jobs, the individual consists of a second list of numbers, where each number denotes the index of the machine on which

the corresponding job should be scheduled. Therefore, it is not required to determine the associations of jobs to machines heuristically, since it is explicitly defined in this encoding. Figure 3 represents the solution by using this encoding. The upper list represents the permutation of jobs, while the lower list denotes the indices of the machines to which the jobs are allocated. The figure shows that the first job which needs to be executed is job $J_6$ and that this job will be scheduled on machine $M_2$ (specified by the second list). The rest of the schedule can be constructed in the same way.

| | | | | | | |
|---|---|---|---|---|---|---|
| 6 | 2 | 5 | 1 | 0 | 4 | 3 |
| 2 | 0 | 1 | 0 | 2 | 0 | 1 |

Figure 3: Schedule represented by PEML

*4.3. Permutation encoding with machine count*

The permutation encoding with machine count (PEMC) is similar to the PEML encoding scheme in a way that it extends the permutation list with an additional integer array Carter & Ragsdale (2006). In this representation, the length of the integer array is equal to the number of machines. Each number in the integer array represents the number of jobs that will be associated with the machine with the corresponding index. This would mean that the first number in the array would represent how many jobs from the start of the permutation would be associated to the machine with index 0, while the next number would denote how many jobs from the remaining permutation would be assigned to the machine with index 1, and so on. The consequence of such an encoding scheme is that the sum of the elements in the integer array needs to be equal to the number of jobs, thus imposing an additional constraint that needs to be satisfied when applying genetic operators. Figure 4 represents the solution using this encoding. The upper list represents the permutation of the jobs, while the lower list represents the integer array that denotes the number of jobs assigned to each machine. The integer array denotes that machine $M_0$ will contain the first three jobs denoted in the permutation list: $J_2$, $J_1$, and $J_4$. Machine $M_1$ will contain the next two jobs, $J_5$ and $J_3$, since the value on index 1 in the integer array is equal to 2. Finally, the last index in the integer array is also equal to 2, which denotes that the next two jobs, which are also the last in the permutation list, will be scheduled on machine $M_2$.

| | | | | | | |
|---|---|---|---|---|---|---|
| 2 | 1 | 4 | 5 | 3 | 6 | 0 |
| 3 | 2 | 2 | | | | |

Figure 4: Schedule represented by PEMC

| 2.53 | 0.31 | 0.12 | 1.97 | 0.81 | 1.43 | 2.28 |
|---|---|---|---|---|---|---|

Figure 5: Schedule represented by RKE

### *4.4. Random key encoding*

The random key encoding (RKE) uses a single list of real numbers to represent the schedule (Bean, 1994; Behnamian et al., 2009). All the numbers in this encoding must be from the range $[0, m >$, and the length of the list is equal to the number of jobs in the scheduling problem. Each of the numbers in the list is associated with the job with the index that corresponds to the position of the number in the list. This means that the number at the first position would be associated with job $J_0$, the number at the second position with job $J_1$, and so on. The number is used to determine both the allocation of this job to a machine and the sequence of the jobs on that machine. The integer part of this number denotes the index of the machine on which the job should be scheduled. The fractional part of the priority value determines the position of the job on the machine in a way that jobs with a smaller fractional value will be scheduled earlier, while jobs with a larger priority value will be scheduled at a later moment in time.

Figure 5 represents the solution encoded by using RKE. The first number denoted in the list corresponds to job $J_0$ and can be used to determine how this job will be scheduled. Since the integer part of this number is equal to 2, job $J_0$ will be scheduled on machine $M_2$. However, it is still required to determine when the job will be scheduled. Since job $J_6$ is the only other job in this list which will be scheduled on machine $M_2$, the fractional parts of these two numbers are compared, and since job $J_6$ has a smaller fractional value, it will be scheduled before job $J_0$ on machine $M_2$. The rest of the schedule for the other two machines is constructed in the same way.

### *4.5. Floating point encoding*

The floating point encoding (FPE) is a representation that uses a list of real numbers, similar to RKE (Đurasević & Jakobović, 2016). However, in this representation, the numbers can have values between $[0, 1]$. Therefore, the information about the machine on which the job needs to be scheduled is also encoded in the fractional part of the number. To determine on which machine the job will be scheduled, the interval $[0, 1]$ is divided into $m$ equal subintervals. The machine on which the job will be scheduled is selected by determining to which of the intervals the priority value of the job belongs. For example, if there are two machines, then the interval for machine $M_0$ would be $[0, 0.5 >$, while the interval for machine $M_1$ would be $[0.5, 1]$. Therefore, if the priority value belongs to the first interval, the job will be scheduled on machine $M_0$. Otherwise, it will be scheduled on machine $M_1$. After the allocation of jobs to machines is determined, the sequence of jobs on each of the machines is determined by arranging them by the values of their priorities in increasing order. This means

that jobs with smaller priority values will be scheduled sooner, while those jobs with a larger priority value will be scheduled towards the end of the schedule.

Figure 6 shows the solution represented by this encoding. Since this problem consists out of three machines, the intervals will be $[0, 0.33 >$ for machine $M_0$, $[0.33, 0.66 >$ for machine $M_1$, and $[0.66, 1]$ for machine $M_2$. By using these intervals it can be determined that jobs $J_1$, $J_2$, and $J_4$ should be scheduled on machine $M_0$, jobs $J_3$ and $J_5$ on machine $M_1$, and jobs $J_0$ and $J_6$ on machine $M_2$. The sequence of these jobs is then determined by their priority values. For machine $M_0$ it is evident that job $J_2$ needs to be scheduled first since it has the smallest priority value out of all the jobs which are scheduled on that machine. After it, job $J_1$ will be scheduled since it has the second smallest value, whereas job $J_4$ will be scheduled the last since it has the largest value out of all three jobs. On machine $M_1$ the first job that would be executed is $J_5$, followed by job $J_3$. Finally, on machine $M_2$ the first job that would be executed would be $J_6$ followed by $J_0$, again because of the reason that $J_6$ has a smaller value assigned to it than $J_0$.

| | | | | | | |
|---|---|---|---|---|---|---|
| 0.85 | 0.21 | 0.13 | 0.57 | 0.29 | 0.44 | 0.71 |

Figure 6: Schedule represented by FPE

### *4.6. Matrix encoding*

The matrix encoding (ME) uses a matrix of real numbers to represent the solutions (Balin, 2011). In this encoding rows denote machines, while columns denote jobs. Thus, the matrix consists of $m$ rows and $n$ columns. Each cell in the matrix contains a real number in the range $[0, 1]$. If the value of the cell is equal to 0, this means that the job with the corresponding column index is not scheduled on the machine with the corresponding row index. On the other hand, if the cell contains a value which is larger than 0, the job is then scheduled to the corresponding machine. The values of the cells specify the sequence in which the jobs will be scheduled on the machines. If the cell contains a smaller value the jobs will be scheduled sooner, whereas jobs with a larger value will be scheduled at a later part of the schedule. When using this encoding, it is required to introduce additional constraints to ensure that the individual always represents a feasible schedule. Therefore, in each column, only one cell can have a value different than 0. If all cells in a column would have the value 0, this would mean that the job would not be scheduled on any of the machines. On the other hand, if more than one cell in a column would have a value larger than 0, this would mean that the job would be scheduled on two or more machines.

Figure 7 shows the solution represented by using ME. For the tested scheduling problem instance, the matrix consists out of three rows (which is equal to the number of machines), and seven columns (which is equal to the number of jobs). By going through each row in the matrix and determining which cells have a value larger than 0, it can be deduced which jobs need to be scheduled on

| Machine index \ Job index | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0.5 | 0.3 | 0 | 0.9 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0.8 | 0 | 0.2 | 0 |
| 2 | 0.7 | 0 | 0 | 0 | 0 | 0 | 0.1 |

Figure 7: Schedule represented by ME

which machine. For example, in the row which represents machine $M_0$, the jobs with indices 1, 2, and 4 have a value larger than zero, meaning that these jobs need to be scheduled on machine $M_0$. However, it is still required to determine the sequence in which they will be scheduled. Since in this example job $J_2$ has the smallest priority value, it will be scheduled first, followed by job $J_1$, and finally by job $J_4$, which has the largest priority value. The rest of the schedule is constructed in the same way.

### *4.7. Machine list encoding*

The machine list encoding (MLE) is a simple solution representation in which each machine has a list that contains the permutation of all jobs scheduled on that machine (Vallada & Ruiz, 2011). This makes the decoding of this list very simple since it is required to traverse the list of each machine and schedule all the jobs in the order in which they are listed. However, when using this representation, it is required to ensure that each job appears in exactly one list, otherwise, the individual would not represent a feasible solution.

Figure 8 represents the solution encoded with this representation. Since the considered scheduling problem consists out of three machines, the solution representation contains three lists of job permutations. The first list denotes which jobs need to be scheduled on machine $M_0$. In this example jobs with indices 2, 1, and 4 would be scheduled in that order on machine $M_0$. For the other two machines, the jobs which need to be scheduled on them and their order can be determined in the same way.

| Machine index | | | |
|---|---|---|---|
| 0 | 2 | 1 | 4 |
| 1 | 5 | 3 | |
| 2 | 6 | 0 | |

Figure 8: Schedule represented by MLE

## 5. Experimental setup

In this paper the following four scheduling problems will be used to test the previously described solution representations: $Rm|r_j|C_{max}$, $Rm|r_j|Ft$, $Rm|r_j|Nwt$, and $Rm|r_j|Twt$. As can be seen from the definitions, four scheduling criteria will be optimised independently on problems that contain jobs that are released over time into the system. Although no additional constraints are considered, some (like setup times) would be also fairly easy to include since they do not influence the solution representation or genetic operators, but only on how the fitness function is calculated.

To test the selected solution representations, a set of 60 problem instances was designed. The design of experiments was performed similarly as in other studies dealing with a similar problem Lin et al. (2013); Lee et al. (2013); Kim et al. (2003). The total fitness of each encoding is calculated as the sum of criteria values of the best solution obtained for solving each problem instance. The problem set consists of instances with different characteristics, to test the GA on various scheduling situations. The problem instances contain between 12 and 100 jobs, as well as between 3 and 10 machines. The properties of jobs in the problem instances are generated in the following way. The processing times of jobs are generated from the interval

$$p_{ij} \in [0, 100],$$

by using one of the following three probabilistic distributions: uniform, normal (Gaussian), and quasi-bimodal. The normal distribution is used with the median value of 50 and a standard deviation of 20, while the quasi bi-modal distribution is defined to have its peaks around values of 15-25 and 80-90. This is done to simulate that jobs from different sources have processing times which differ from each other. It should be noted that these distributions can lead to scenarios that can be considered unrealistic since the job execution times will not have any correlation to the machine that is selected for that execution. All job weights are generated uniformly from the interval

$$w_j \in< 0, 1].$$

The release times of jobs are generated by using a uniform distribution from the interval

$$r_j \in \left[0, \frac{\hat{p}}{2}\right],$$

where $\hat{p}$ is defined as

$$\hat{p} = \frac{\sum_{j=1}^{n} \sum_{i=1}^{m} p_{ij}}{m^2}.$$

The due dates of jobs are also generated by using a uniform distribution from the interval

$$d_j \in \left[r_j + (\hat{p} - r_j) * \left(1 - T - \frac{R}{2}\right), r_j + (\hat{p} - r_j) * \left(1 - T + \frac{R}{2}\right)\right],$$

where $T$ represents the due date tightness parameter, while $R$ represents the due date range parameter. The due date range defines the dispersion, while the due date tightness adjusts the amount of jobs that will be late. While generating the problem set, both of those parameters assumed values of 0.2, 0.4, 0.6, 0.8, and 1 in various combinations.

To improve the performance of each encoding, the parameters of the GA were fine-tuned for each of them. Table 2 denotes the values of all algorithm parameters that were used by the GA for the different solution representations. The list of the genetic operators that were used by the different encodings is:

- Floating point
  - Crossover operators (for brevity let $a$ denote a gene from the first parent, $b$ a gene from the second parent, and $c$ the value of the child gene)
    - Arithmetic - $c = (\alpha) * a + (1 - \alpha) * b$, where $\alpha \in [0, 1]$ and has the same value for all genes in the individual
    - Average - $c = (a + b)/2$
    - BGA - generates an offspring by moving from the better parent in a specified direction, which is calculated based on the worse individual and certain random values
    - BLX-alpha - samples the new value from the interval of the two parents, however, the interval is increased in both directions by the factor alpha
    - Discrete - for each gene it is randomly selected whether it will be inherited from the first or the second parent
    - Flat - the value of the child gene $c$ is randomly generated from the interval $[a, b]$
    - Heuristic - for each gene a linear interpolation between the values in the two individuals is performed, but the values closer to the better parent have a higher chance of being selected
    - Local - the same as the arithmetic crossover, but the $\alpha$ value is randomly generated for each gene
    - onepoint - selects a random point in the individuals and takes all the genes before this point from one parent, and all genes after this point from the other parent
    - SBX - simulates the effect of the one point binary crossover
  - Mutation operators
    - simple - selects a random gene in the individual and samples a new value for it from the domain
- Permutation
  - Crossover operators

    * COSA - performs a 2-op mutation on the first parent, but uses the second parent to determine which new element should be inserted
    * DPX - detects common sub-paths of the two parents and tries to reconnect them to produce the child
    * OBX - random genes are copied from one parent to the child, while the remaining genes are filled out from the other parent starting from the beginning of the individual
    * OPX - selects a random crossover point, and copies all elements up to that position from the first parent, and fills out the rest from the second parent
    * OX - a part of one parent is copied into the child, whereas the missing numbers are copied from the other parent in the order in which they appear in it, but starting from the end of the copied interval
    * OX2 - the same as OX, but the genes from the second individual are copied from the start of the individual
    * PBX - similar to OX, however, instead of copying a connected sequence of genes from the first parent, random genes are selected and copied from the parent to the corresponding positions of the child
    * PMX - a part of one parent is copied into the child, while the rest of the individual is filled out from the second parent, partially by using a mapping from the transferred part, and partially by just being copied over
    * SPX - two positions in both individuals are selected and the elements at the chosen places are swapped
    * ULX - each gene is randomly selected from one of the parents, and the missing entries are then filled randomly at the end
    * UPMX - similar to PMX, but instead of using the crossover points to determine the swaps, they are considered for each position
    * Cyclic - creates a number of cycles by using both parents, and then copies each of the cycles alternatively from each parent to the child until all the cycles are copied
  - Mutation operators
    * Insert - selects a random element and inserts it at a random position in the individual
    * Inverse - selects two random positions and reverses the order of the elements between them
    * Toggle - selects two random positions and swaps their values

- Integer genotype

  - Crossover operators
    * Uniform - each gene in the child is selected randomly from one of the parents
  - Mutation operators
    * Simple - the value of a random gene is replaced by a random value generated from the allowed domain

- Integer genotype (with sum constraints)
  - Crossover operators
    * Crx1x - a random point in the parents is selected and the part before the point is copied from the first parent, while the second part is copied from the second parent. A repairing procedure, which increments or decrements randomly selected genes, is applied until the sum of the elements satisfies the required condition
    * Crx0x - each element of the child is randomly selected from one of the parents. The repairing procedure is the same as in Crx1x
  - Mutation operators
    * Leveling - selects the largest number in the individual and decrements it, after which it selects the smallest number and increments it
    * LevelingR - same as Leveling, but the amount by which the numbers are decremented and incremented is a random number between 0 and the largest value in the individual
    * MaxR - finds the largest element, decrements it by one, and increments the value of the element to the right
    * MaxRR - same as MaxR, but the number by which the elements are decremented and incremented is a random number between 0 and the largest value in the individual
    * MaxRandom - selects the largest value in the individual, decrements it and then increments one other randomly selected value
    * MaxRandomR - same as MaxRandom, but the number by which the elements are decremented and incremented is a random number between 0 and the largest value in the individual
    * Randomx2 - selects two random elements, one of which is decremented and the other is incremented
    * Randomx2R - same as Randomx2, but the number by which the elements are decremented and incremented is a random number between 0 and the value of the element which is decremented

- Matrix genotype
  - Crossover operators

        * Uniform - each column of the matrix is randomly selected from one of the parents
    - Mutation operators
        * Cell swap - a random cell with a value different than 0 is selected and is swapped with a different cell in the same column
        * Cell value - a new random number is generated for the cell from the allowed interval

- List genotype
    - Crossover operators
        * Point - for each list a crossover point is selected, and for each lists the elements before that point are taken from the first parent. The missing elements are then taken from the second parent.
    - Mutation operators
        * Shift - a job is randomly selected and shifted to a different position in the same list
        * Insert - a job is randomly selected and inserted into another randomly selected list

If more than one operator is defined for a certain representation, then one of the available operators will be randomly selected and used in each iteration. Furthermore, for PEML which consists out of two genotypes, the operators are applied independently on each genotype. However, in each iteration, the operator will be applied only on one of the genotypes. More details about the genetic operators that were used for the various solution representations, can be found in the documentation of the Evolutionary Computation Framework (ECF) in which all the implementations were performed [1].

Table 2: Parameters used by the different representations

| Parameter | PE | PEML | PEMC | RKE | FPE | ME | MLE |
|---|---|---|---|---|---|---|---|
| Population size | 30 | 30 | 1000 | 30 | 30 | 30 | 30 |
| Mutation probability | 0.7 | 0.7 | 0.7 | 0.9 | 0.3 | 0.9 | 0.9 |
| Selection | Steady state tournament selection with tournament size of 3 individuals | | | | | | |
| Termination condition | 1 000 000 function evaluations | | | | | | |
| Operators | permutation based operators | permutation and integer based operators | permutation and integer (with sum constraint) based operators | floating point based operators | floating point based operators | matrix based operators | list based operators |

[1] Evolutionary Computation Framework: http://ecf.zemris.fer.hr/

To obtain statistically significant results, the GA was executed 30 times to optimise the set containing all problem instances. From each run, the best individual for each problem instance was saved, and based on those values the minimum, median, and maximum fitness values were calculated for all 30 executions. In addition to those three values, the total minimum value, denoted as *Tmin*, was also calculated. This value represents the minimum value which would be obtained if the best solution for each problem instance would be selected out of all the 30 algorithm executions. This value serves as an indicator of how good solutions the encodings would be able to obtain, given that they would be executed several times for each problem instance. To test whether the results obtained by the methods are significantly different from each other, the Mann-Whitney statistical test was used. The results are considered statistically significant if $\alpha < 0.05$.

## 6. Results

The results obtained by the tested solution representations are denoted in Table 3. For each row, the solution representation which achieved the best result is denoted with a grey colour. To better denote the obtained results, they are additionally represented as box plots shown in Figure 9. In the box plot, an additional rhomboid is used to denote the average value, to outline whether there is a large difference between it and the median value.

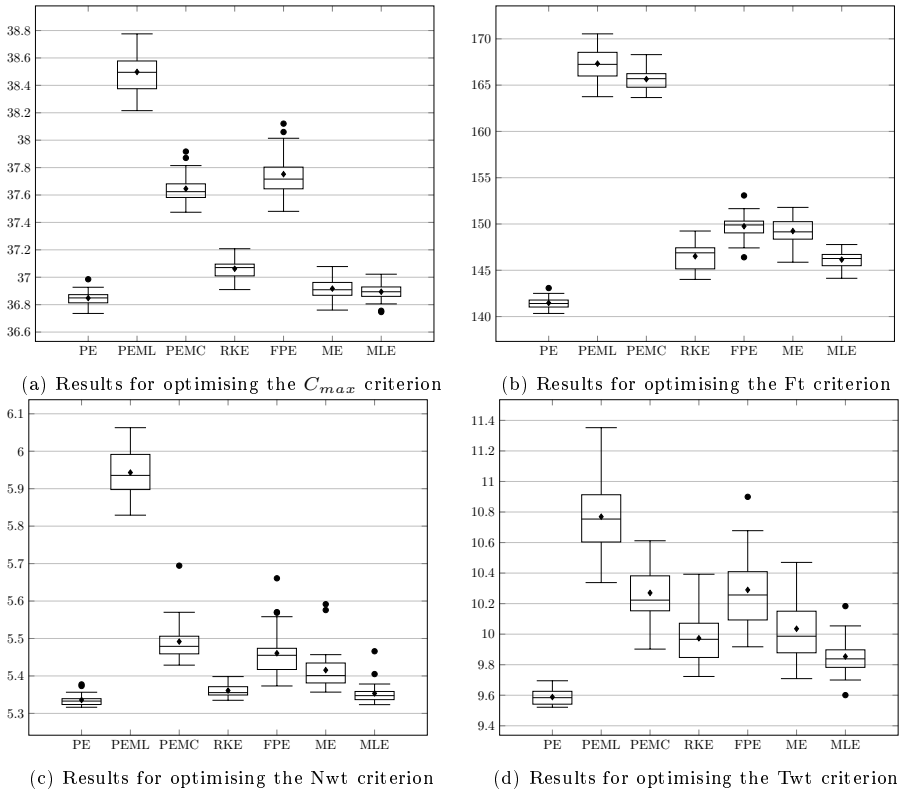
The table denotes that the GA obtained the best results when using PE for all four tested criteria. This encoding has proven to be especially efficient when optimising the $Twt$ and $Ft$ criteria since for those two it achieved the best improvements when compared to the other encodings. Furthermore, from the box plot, it is also evident that the GA achieved the least dispersed results when using this encoding. Such a behaviour is desirable since it denotes that the algorithm behaves quite stable and will achieve mostly similar results even upon several invocations. Extending this encoding with an additional list of integer numbers, which denote the allocation of jobs to machines, has not proven to be beneficial since the GA achieved quite poor results when using PEML. For this encoding, the GA even achieved the worst results out of all the tested solution encodings, which consistently happens for all four optimised scheduling criteria. Furthermore, the GA achieved quite dispersed results when using this representation, which denotes that for several invocations of the algorithm it is expected that quite different results will be obtained. PEMC, the alternative representation which uses an additional integer array, achieves generally much better results than PEML. For the $Twt$ and $C_max$ criteria, it even outperformed one of the floating point representations. However, when compared to the other representations, it still achieved inferior results.

By using the solution representations based on a single array of real numbers, the GA performs worse than by using PE but much better than when using PEML. However, the GA does not perform equally well for both of the encodings. When using FPE, the GA usually achieved worse results than when using any of the other encodings, except for PEML and PEMC. This can quite

Table 3: Results obtained by the tested solution representations

| Criterion | | Solution encoding | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | PE | PEML | PEMC | RKE | FPE | ME | MLE |
| Execution time | | 681.0 | 508.1 | 485.7 | 436.8 | 339.1 | 489.5 | 505.4 |
| $C_{max}$ | Tmin | 36.59 | 37.14 | 36.64 | 36.55 | 36.79 | 36.50 | 36.52 |
| | Min | 36.74 | 38.22 | 37.47 | 36.91 | 37.48 | 36.76 | 36.75 |
| | Med | 36.85 | 38.50 | 37.62 | 37.07 | 37.72 | 36.91 | 36.90 |
| | Max | 36.99 | 38.78 | 37.92 | 37.21 | 38.12 | 37.08 | 37.02 |
| $Ft$ | Tmin | 138.9 | 151.2 | 150.74 | 139.7 | 140.8 | 141.0 | 139.7 |
| | Min | 140.3 | 163.8 | 163.7 | 144.0 | 146.4 | 145.9 | 144.1 |
| | Med | 141.4 | 167.3 | 165.7 | 146.9 | 149.9 | 149.2 | 146.3 |
| | Max | 143.1 | 170.5 | 168.9 | 149.2 | 153.1 | 151.8 | 147.8 |
| $Nwt$ | Tmin | 5.316 | 5.615 | 5.364 | 5.334 | 5.340 | 5.317 | 5.318 |
| | Min | 5.316 | 5.829 | 5.429 | 5.335 | 5.373 | 5.357 | 5.323 |
| | Med | 5.333 | 5.938 | 5.480 | 5.356 | 5.455 | 5.403 | 5.348 |
| | Max | 5.377 | 6.063 | 5.694 | 5.398 | 5.661 | 5.592 | 5.466 |
| $Twt$ | Tmin | 9.452 | 9.725 | 9.585 | 9.516 | 9.533 | 9.525 | 9.499 |
| | Min | 9.521 | 10.34 | 9.902 | 9.723 | 9.917 | 9.709 | 9.601 |
| | Med | 9.584 | 10.76 | 10.24 | 9.968 | 10.27 | 9.987 | 9.846 |
| | Max | 9.695 | 11.35 | 10.61 | 10.39 | 10.90 | 10.47 | 10.18 |

easily be seen when optimising the $Twt$ and $C_{max}$ criteria, where this encoding did not perform well when compared to the others. The dispersion of the obtained solutions is larger than when using RKE, or some other solution representations. On the other hand, when using RKE the GA achieves better results. This encoding has proven to be more suitable for some criteria than for others. For the $Ft$ and $Nwt$ criteria the GA obtained the third best results when using this encoding. It can even be seen that the differences in results between this encoding and the results obtained by the second-best encoding are similar.




(a) Results for optimising the $C_{max}$ criterion

(b) Results for optimising the Ft criterion

(c) Results for optimising the Nwt criterion

(d) Results for optimising the Twt criterion

Figure 9: Comparison of the results obtained by the different solution representations

When using ME, the performance of the GA depends on the criterion which is optimised. Furthermore, this representation also obtained more dispersed results than the other three aforementioned solution representations. The final encoding, MLE, consistently obtained good results for all four tested scheduling criteria. The GA achieved the second-best median values of the results for all the scheduling criteria when using this encoding. It was outperformed only by the GA which used PE. Although for some optimisation criteria this encoding achieved quite similar results as others (like ME for the $C_{max}$ criterion, or RKE for the $Ft$ criterion), neither of the other encodings consistently achieved such good results as this one. An additional benefit of this encoding is that the GA

Table 4: Results of the statistical tests

(a) Results for the $C_{max}$ criterion

| | PE | PEML | PEMC | RKE | FPE | ME | MLE |
|---|---|---|---|---|---|---|---|
| PE | - | < | < | < | < | < | < |
| PEML | > | - | > | > | > | > | > |
| PEMC | > | < | - | > | < | > | > |
| RKE | > | < | < | - | < | > | > |
| FPE | > | < | > | > | - | > | > |
| ME | > | < | < | < | < | - | = |
| MLE | > | < | < | < | < | = | - |

(b) Results for the $Ft$ criterion

| | PE | PEML | PEMC | RKE | FPE | ME | MLE |
|---|---|---|---|---|---|---|---|
| PE | - | < | < | < | < | < | < |
| PEML | > | - | > | > | > | > | > |
| PEMC | > | < | - | > | > | > | > |
| RKE | > | < | < | - | < | < | = |
| FPE | > | < | < | > | - | = | > |
| ME | > | < | < | > | = | - | > |
| MLE | > | < | < | = | < | < | - |

(c) Results for the $Nwt$ criterion

| | PE | PEML | PEMC | RKE | FPE | ME | MLE |
|---|---|---|---|---|---|---|---|
| PE | - | < | < | < | < | < | < |
| PEML | > | - | > | > | > | > | > |
| PEMC | > | < | - | > | > | > | > |
| RKE | > | < | < | - | < | < | > |
| FPE | > | < | < | > | - | > | > |
| ME | > | < | < | > | < | - | > |
| MLE | > | < | < | < | < | < | - |

(d) Results for the $Twt$ criterion

| | PE | PEML | PEMC | RKE | FPE | ME | MLE |
|---|---|---|---|---|---|---|---|
| PE | - | < | < | < | < | < | < |
| PEML | > | - | > | > | > | > | > |
| PEMC | > | < | - | > | = | > | > |
| RKE | > | < | < | - | < | = | > |
| FPE | > | < | = | > | - | > | > |
| ME | > | < | < | = | < | - | > |
| MLE | > | < | < | < | < | < | - |

obtains results that are not as heavily dispersed as those obtained by some other encodings.

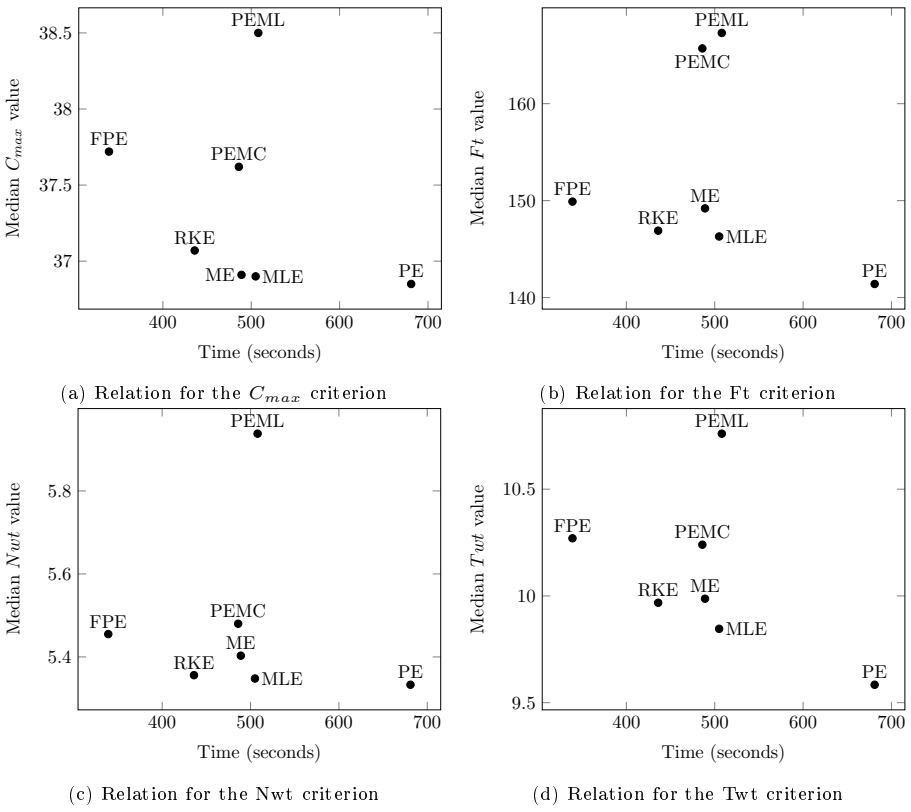
To test which of the encodings lead to significantly better results, statistical tests were performed between the results obtained by the encodings for all four optimised scheduling criteria, and are presented in Table 4. The tables denote whether the results obtained by the encodings denoted in rows of the table are statistically different from the results obtained by the encodings denoted in the columns of the table. Therefore, cells that contain "<" specify that the encoding denoted in the row achieves significantly better results than the encoding denoted in the column. Cells that contain ">" specify that the encoding denoted in the row achieves significantly worse results than the encoding denoted in the column. Finally, cells that contain "=" denote that there is no significant difference between the encodings denoted in the row and column. Additionally, cells in which the encoding denoted in the row performs significantly better than the encoding denoted in the column are shaded with grey colour to better denote the best encodings. The statistical tests demonstrate that GA with PE obtained significantly better results than any of the other encodings, for all four optimised criteria. On the other hand, for PEML the GA obtained significantly worse results than by using any of the other encodings. MLE obtains significantly better results than any of the other encodings (except for PE) for all but two situations, where it achieves equally good results as ME and RKE.

Apart from the value of the optimised criteria, the execution time of the algorithm for each of the encodings also represents an important factor. Table 3 denotes the execution time (denoted in seconds) of each algorithm for all the tested problem instances. It can immediately be seen that different encodings have large differences between their execution times. The shortest execution time of the algorithm is achieved when using FPE. On the other hand, the GA executes the slowest when using PE, and is approximately two times slower than by using FPE. From the results, it is evident that the type of encoding which is used will have a large influence on the execution time of the algorithm. For

example, the encodings which use floating point numbers to represent solutions, namely RKE and FPE, lead to a smaller execution time of the algorithm, while the encodings which use the list of permuted jobs, like PE, PEML, PEMC, and even MLE, will lead to larger execution times.

The execution times are also plotted together with the median values of the optimised criteria in Figure 10. This figure shows how the encodings are ranked if both the quality of the solutions and the execution time of the GA are of equal importance. The behaviour which occurs for all criteria is that the best performance, but also the slowest execution time, is obtained when using PE. On the other hand, the lowest execution time, but not the worst performance, is obtained when using FPE. All the other encodings provide a different trade-off between the execution time and the performance, usually in a way that if they obtain a better performance they also have a larger execution time. However, there are cases in which an encoding is outperformed for both criteria by another encoding, making it redundant in that case. For example, this is the case with PEML, since it is always outperformed by all other encodings (except for PE) for both the execution time and the quality of the constructed schedule. Therefore, selecting PEML would not be beneficial in any way. A similar situation occurs for the PEMC encoding since the RKE encoding always achieves a better result in less time. The results obtained by using ME are always outperformed by RKE for both the execution time and the quality of the constructed schedules for all criteria except $C_{max}$, making this encoding uncompetitive for three scheduling criteria. On the other hand, RKE and MLE provide the same trade-off for all four criteria, with RKE having a smaller execution time, but with MLE obtaining better values for the optimised criteria. Therefore, when taking both the quality of the schedule and the execution time of the algorithm into account it is not as easy to select the best encoding.

Besides testing the methods with a fixed number of function evaluations, it is also interesting to analyse how the encodings perform if a fixed time limit for each problem instance is imposed. Therefore, for each problem instance, the GA will receive a time limit of five seconds to find the solution for it, and the change of the fitness values on all problem instances in each moment in time will be represented in Figure 10. For each criterion, an additional zoomed-in subfigure is included to better denote the performance of the encodings, in situations when they achieve a similar performance. The first thing that can be noticed is that for PEML and PEMC the GA almost consistently achieves the worst results during the entire execution time, although in the end PEMC does achieve slightly better results. For PE the GA exhibits a very interesting behaviour. Except for the $C_{max}$ criterion, the results obtained for this encoding are quite poor at the beginning of the evolution process. However, during the execution of the algorithm, the fitness value for this encoding decreases faster than when using other encodings. For three out of the four optimised criteria, this encoding obtains the best results in the end. Between the other four encodings, there is not such a large difference between the values of their starting solutions. It is interesting to observe that in some cases the GA with FPE had a better start than when using RKE, but RKE leads to better convergence, and in the




(a) Relation for the $C_{max}$ criterion

(b) Relation for the Ft criterion

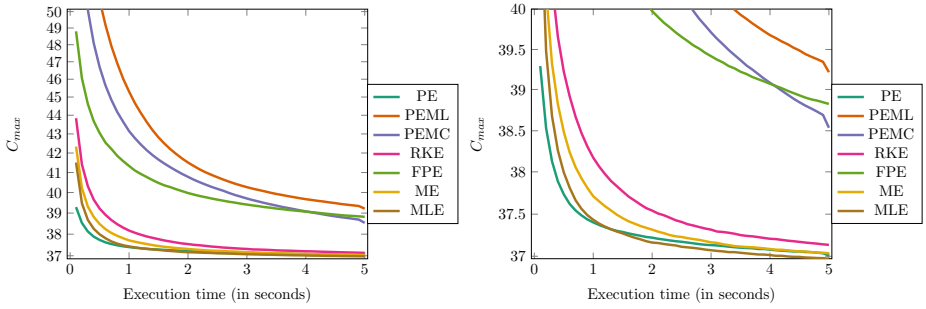
(c) Relation for the Nwt criterion

(d) Relation for the Twt criterion

Figure 10: Relation between the execution time and the performance of the GA when using various representations

end obtained superior results. By using MLE the GA usually started with the best solutions, which in the end had a good effect on the final performance of the algorithm, since it was never outperformed by any other encoding except for PE. By using ME the GA also obtains good results at the start of the evolution process. However, it is evident that the results do not improve much, and therefore this encoding is usually easily outperformed by RKE in the later iterations. For the $Ft$, criterion even FPE can obtain a better performance than ME for some time, but in the end, ME was able to slightly outperform FPE, although with great difficulty.

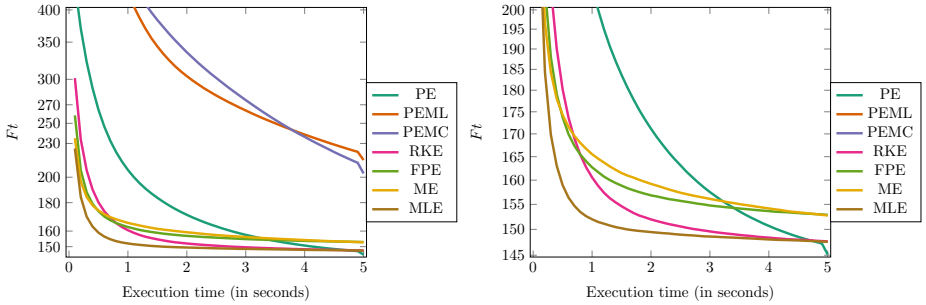
To provide a more detailed analysis of each encoding, two additional experiments were performed. In the first experiment, the goal was to compare the evaluation times of each encoding. Therefore, 100 000 random individuals were evaluated for each problem instance. In that way, the complexity of each decoding scheme can be approximated. In the second experiment, the goal was to obtain a notion of the complexity of genetic operators used for different encoding schemes. This experiment used only three individuals in the population with all the genetic operators that were outlined before. Furthermore, the mutation probability was set to 1 to ensure that in each iteration both the crossover and mutation operators are applied, while the fitness function was modified to return a fixed value so that no time is spent on evaluation of the individual. This experiment was executed until one million function evaluations were performed. Table 5 represents the obtained results for both experiments. As can be seen, depending on the representation, the evaluation of the individual can be quite different. The PEML, PEMC, and MLE encodings have the lowest evaluation time, which is expected since they are easy to decode and evaluate. The PE and ME encoding require roughly two times more time to be evaluated than the previous two encodings. For PE this is a consequence of the fact that the solution is constructed by a heuristic procedure. On the other hand, although in ME the allocation of jobs to machines can be easily determined, their order on the machines needs to be decoded out of the priorities assigned to each job. Finally, the encodings based on the floating point representation require the most time to be decoded. This is a consequence of the numeric operations that need to be performed to extract the machine allocation and job sequence out of the real numbered values. As far as the genetic operators are concerned, the situation is slightly different. The least amount of time needed to perform the genetic operators is required by the floating point representations. This was expected since these operators are based on simple mathematical operations that can be performed quickly. The most time to perform the operations is required by the operators for the permutation representation, as well for the custom operators for the MLE and ME. This happens since the implementation of the operators is more complex, because the operator has to ensure that it will generate a valid individual. One thing which might seem surprising at first is that the PE encoding has a higher execution time than the PEMC encoding. However, this is the consequence of the fact that when two genotypes are used at the same time, only one of them will be modified in each iteration, thus reducing the amount of time that the permutation operators will be performed on the individual.




(a) Results for the $C_{max}$ criterion



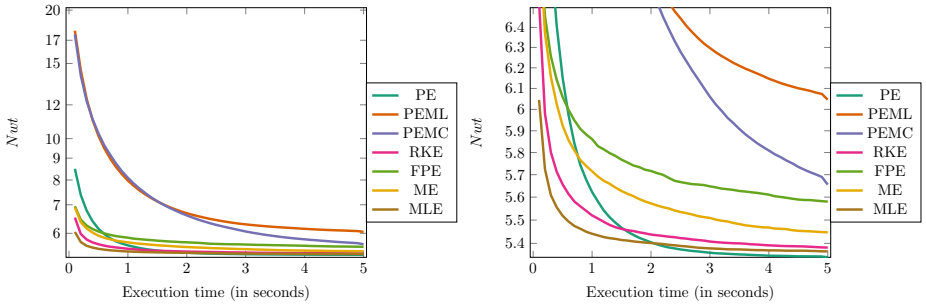

(b) Results for the $C_{max}$ criterion (zoomed-in)




(c) Results for the $Ft$ criterion



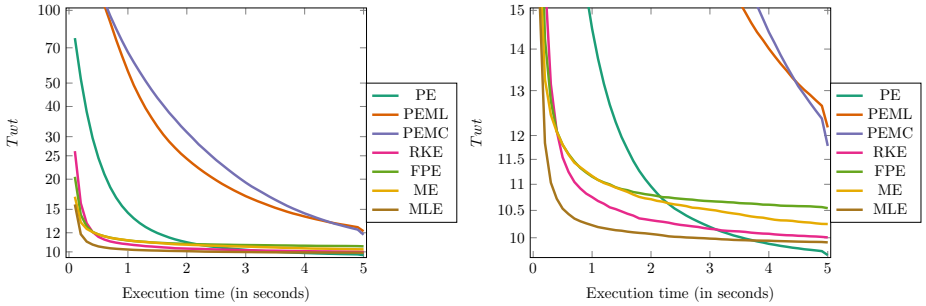

(d) Results for the $Ft$ criterion (zoomed-in)




(e) Results for the $Nwt$ criterion




(f) Results for the $Nwt$ criterion (zoomed-in)




(g) Results for the $Twt$ criterion




(h) Results for the $Twt$ criterion (zoomed-in)

Figure 11: Change of the minimum fitness value during the execution of the algorithm

Table 5: Influence of the encoding on the evaluation time and operator execution time

| | PE | PEML | PEMC | RKE | FPE | ME | MLE |
|---|---|---|---|---|---|---|---|
| Evaluation time | 25.86 | 12.51 | 12.49 | 34.06 | 28.00 | 24.18 | 16.61 |
| Operator execution time | 443.7 | 538.4 | 398.0 | 236.4 | 236.5 | 359.4 | 419.6 |

## 7. Discussion

This section will provide a short discussion based on the results and findings from the previous section. Table 6 represents an overview of the most important characteristics of each encoding. The *rank* column denotes the total rank for all the optimised criteria by each encoding, which was calculated based on the median values obtained by the GA. A lower rank for an encoding denotes that the GA obtained better values for the criteria when compared with other encodings. For example, a rank of 1 would denote that for this encoding the GA obtained the best median value for all four criteria. The *execution time* column denotes the total execution time for all encodings when the maximum number of function evaluations specified in Table 2 is used. The *space* column denotes the memory space which is required by each of the encodings to represent a single solution. The *complete* column denotes whether by using the encoding, it is possible to represent all existing solutions for a certain problem. The *default operators* column denotes whether a standard solution representation was used for which standard genetic operators already exist so that it is not required to define specialised operators for this encoding. The *unique* column denotes whether any two encoded solutions always represent two distinct schedules. This means that each schedule can be mapped to only a single encoded solution. Finally, the "interpretable" column denotes whether the solution represented by a certain encoding is easy to interpret by humans. Although classifying the encodings by this property is quite subjective, the main criterion which was used is whether it is easy for a human to determine the allocations of jobs to machines and their sequence on them, without having to decode the encoded solution.

Based on its rank, PE achieved the best results on all four scheduling criteria. Even more, the statistical tests have demonstrated that the results obtained by this encoding are significantly better than the results obtained by any of the other encodings. Thus, if only the quality of the results is of the essence, then this encoding can be considered superior to all the other tested encodings. Therefore, specifying only the sequence in which jobs should be scheduled seems to have a quite good influence on the GA, since it has to focus only on finding the right sequence of jobs, whereas the simple heuristic proves to be more than enough to determine a good allocation of these jobs on the available machines. The change of fitness during the evolution process outlines a very interesting behaviour of the GA when using this encoding. At the start of the evolution process, the GA requires a certain amount of time to obtain good solutions. Thus, it seems to be difficult for the algorithm to obtain a good permutation of jobs. However, as soon as a good solution is obtained, it can be seen that

Table 6: Properties of the tested solution representations

| Encoding | Rank | Execution time | Space | Complete | Default operators | Unique | Interpretable |
|---|---|---|---|---|---|---|---|
| PE | 1 | 681.0 | n | No | Yes | No | No |
| PEML | 7 | 508.1 | 2*n | Yes | Yes | Yes | Yes |
| PEMC | 5.5 | 485.7 | n+m | Yes | No | Yes | Yes |
| RKE | 3.25 | 436.8 | n | Yes | Yes | No | No |
| FPE | 5.5 | 339.1 | n | Yes | Yes | No | No |
| ME | 3.75 | 489.5 | m*n | Yes | No | No | No |
| MLE | 2 | 505.4 | n+m | Yes | No | Yes | Yes |

the GA can quickly improve it and obtain some of the best solutions. An additional benefit of this encoding is that very little space is required to represent a solution since only a list of $n$ elements is needed. Finally, since the solution is simply encoded as a permutation, many existing genetic operators exist and it is not required to define new ones. However, the simple heuristic that is used to determine the allocation of jobs on machines has a large influence on the execution time. Namely, when using a fixed number of function evaluations, the GA obtained the longest execution time for this encoding, which is usually quite larger than the execution times obtained for the other encodings. Nevertheless, if the maximum allowed execution time is specified for each of the encodings, it was evident that, in the given time, this encoding achieved the best results for three criteria, although the obtained improvements in the results were not large in comparison with the second-best encoding. One additional drawback of this encoding is that two different individuals can be used to represent the same solution, which comes as a consequence of the fact that the allocation of jobs to machines is not specified. Therefore, a change in the order of two jobs will not necessarily lead to a different schedule if both jobs are scheduled on different machines. Finally, it is quite difficult to immediately determine the schedule based on this encoding, since the allocation of jobs to machines must be determined heuristically.

For PEML the GA obtained the worst rank, meaning that it performed quite poorly for optimising all scheduling criteria. Therefore, adding an additional genotype to the individual, which represents the allocation of jobs to machines, has resulted in a serious degradation of the performance. The reason for this seems to be twofold. The first one is that by adding the additional genotype the solution space has increased, making it harder to find good solutions. The other reason is that the genetic operators are performed independently on both genotypes. This can lead to situations in which a good permutation of jobs is obtained, but the allocation of jobs on machines is quite poor. Such a problem could be solved by defining genetic operators that would perform changes on both genotypes at the same time. Adding an additional list to the individual increases the space required by the individuals to represent solutions in comparison with PE. On the other hand, by using the additional list it is not required to use a heuristic to allocate the jobs to machines. This leads to an easier interpretation of this encoding. However, when using this solution rep-

resentation the GA still has a longer execution time in comparison with other encodings except for PE. Another benefit of this encoding is that it can represent all possible solutions. Also, since the encoding uses one permutation and one integer genotype, it is not required to define any new operators. Finally, every change which happens in the encoded solution will necessarily result in a different schedule. For example, a change in the permutation genotype will lead either to a different sequence of the jobs on a machine or to the placement of the job on another machine, while a change in the integer genotype will lead to a different allocation of a job to a machine.

The alternative PEMC encoding, which also uses an additional integer array, achieves significantly better results than the PEML encoding. Therefore, it seems easier for the algorithm to work on a more concise representation since here the integer array will be much shorter than in PEML. One possible reason for this is that when jobs that are close to each other in the permutation array are swapped, they will mostly remain on the same machine. This makes it easier for the algorithm to find a good permutations that should be allocated to each machine. This is an interesting result because one might have expected PEML to obtain better results simply because it offers more diversity of individuals. This representation has even shown to be competitive to FPE. On two of the tested criteria, it achieved better results than FPE, while on the other two it was inferior. Regarding the execution time, the GA had the shortest execution time when using this encoding out of all the permutation-based encodings. This encoding even achieves a slightly better execution time when compared to the ME and MLE encodings, and is only outperformed by the floating point based encodings in that regard. The benefits of this encoding are that all solutions can be represented, that it needs only a slightly larger space than PE (usually the number of jobs is larger than the number of machines), and the solution can be interpreted relatively easily. Furthermore, each change in the genotype will also affect the solution, meaning that the same solution cannot be represented by two different individuals. On the other hand, a disadvantage of this encoding is that for the integer genotype specialised operators, which ensure that all the numbers in the array sum up to a certain value, need to be defined.

By using RKE the GA usually obtained the third best results, which can be seen from the rank of this encoding. Only for the $C_{max}$ criterion, the GA achieved the fourth best result for this encoding. Thus, encodings based on real numbers also seem to be quite well suited for representing solutions in scheduling problems. This result is a bit surprising since the genetic operators which are applied to the individuals do not have any knowledge about the underlying problem, and thus can introduce many changes in the solutions at the same time, or can introduce changes which do not affect the underlying solution. However, this does not seem to significantly influence the performance of the GA. Besides the fact that many existing genetic operators exist which can be used for this encoding, it has several other benefits. When using this solution representation the GA obtained the second best execution time, meaning that the decoding of solutions and the entire evolution process are cheaper than when using other encodings. Furthermore, the entire schedule is encoded by

a single list of real numbers, which makes it one of the most memory efficient encodings. By using this encoding it is also possible to represent all solutions, therefore the entire search space can be represented. On the other hand, one issue associated with this encoding is that small changes to an encoded solution will not necessarily lead to a different schedule. For example, the fractional part of a number may be changed only slightly, which does not cause any change to the sequence in which the jobs are scheduled. This can happen more often on problems that contain a smaller number of jobs since then larger changes in the fractional part will be required to invoke a change in the decoded solution. Thus, a single schedule can be represented by several encoded solutions. Furthermore, this encoding is not easily interpretable. Even though the allocation of jobs to machines can easily be determined, the sequence of jobs is not as evident from the encoding, especially for a larger number of jobs.

The second encoding which is based on real numbers (FPE) has not proven to be as successful as RKE. By using this encoding the GA constantly achieved poor results, and the GA obtained consistently worse results only when using PEML. The cause for this could be because both the allocation and the sequence of jobs are determined only based on the fractional parts of the number. Therefore, a small change in the fractional part can lead to both the allocation of a job on another machine, but also a different position of the job in the sequence. Also, if there are many machines on which the jobs can be scheduled, then the intervals for each machine will be quite small, making it harder to allocate a job on a concrete machine. This kind of encoding seems to be the least computationally expensive since the GA achieved the fastest execution time when using it. Furthermore, as was the case with RKE, this encoding is memory efficient, it can be used to represent all the solutions, and provides various existing genetic operators that can be used. However, even when using this solution representation each schedule will not be represented by only one solution. As was the case with FPE, there will again be several encoded solutions that represent the same schedule. Furthermore, this encoding is even less interpretable than RKE, since even the allocation of jobs to machines is now harder to determine.

ME, which uses a matrix to represent the schedule, usually obtained the fourth best results for the optimised criteria, except for the $C_{max}$ criterion for which it obtained the third best results. When compared with the other encodings, the GA obtains average results when using ME. The reason why ME does not perform as well as for some other representations could be because very simple genetic operators were used for this encoding. For example, the applied uniform crossover operator only selects which part of the solution will be taken from one parent, and which from the other. Therefore, no changes will occur on the priority values of the individual jobs during the crossover, but only during the mutation, which could be the cause for a slower convergence and the poorer results. The only real benefit of this encoding is that it can represent all the solutions, whereas in all other properties it is not very competitive with the other representations. The execution time for this encoding is quite similar to those of PEML and MLE, meaning that it does not provide any significant benefit in this regard since it only significantly outperforms the execution time

of PE. Regarding the space consumption of this encoding, it is the encoding which uses up the most space per individual, since it uses a $m * n$ matrix. Since a matrix is used to represent the solutions, there is not a wide range of operators that can be used, but rather new genetic operators need to be defined, which also need to ensure that no illegal solution is generated during the evolution process. Although this encoding also enables that the allocation of jobs to machines can be easily determined, the sequence of their execution is once again difficult to determine if a large number of jobs are present. Finally, since the sequence of the jobs is again denoted by using real numbers, it is possible to represent the same schedule with more than one encoded solution.

The final solution representation, MLE, leads the GA to obtain quite good results. With this encoding, the GA obtained the second best performance for all four optimised scheduling criteria. This representation can also be seen as a variant of PE since it is still based on using permutations of jobs. However, instead of using only one permutation list, $m$ permutation lists are used, one for each machine. Although this did result in the degradation of the performance to a small extent, it did improve the execution time of the GA when compared to PE. However, the GA with MLE is still much slower than when using either RKE or FPE. The space complexity of this encoding can be considered slightly larger than that of PE since $m$ permutation lists are used to store the permutations for each of the jobs. Furthermore, with this encoding, it is possible to represent all the solutions. An additional benefit is that each encoded solution represents a different schedule, meaning that each change in the solution will lead to a different schedule. Aside from that, this encoding can probably be considered the easiest to interpret, since not only the allocation of jobs to machines is explicitly encoded, but also the sequence of jobs on those machines can easily be determined from the encoded solution. On the downside, no standard genetic operators exist for such an encoding, therefore it is required to implement them. Even though the implemented genetic operators were quite simple and similar to those used by ME, by using them the GA nevertheless obtained solutions of good quality.

## 8. Conclusion

In this paper, seven solution representations have been tested for solving problems in the unrelated machines environment with job release times. All the representations were used for optimising four scheduling criteria to test how they perform on each of them. Aside from testing the performance of all the encodings, the execution time of the GA when using each one of them was also analysed. Furthermore, the encodings were also compared based on some additional properties like space complexity or availability of standard genetic operators.

Based on the obtained results it can be concluded that the best results are obtained when using PE. The statistical tests have shown that the results obtained by this representation are significantly better than those obtained by any of the other encodings. And even though when using a fixed number of

function evaluations the GA achieved a large execution time, by using a fixed execution time this encoding has still demonstrated to obtain the best results for three out of the four optimised scheduling criteria. The only drawback of this encoding is that it can not represent all the possible solutions to a certain scheduling problem. From the other encodings MLE has demonstrated to be the second best one, and also provided quite good results. Thus it can be regarded as a good alternative to the PE encoding. PEML, FPE, and PEMC have shown to be the least competitive, while RKE and ME provided average performance with RKE usually providing better results.

As demonstrated in this paper, the choice of the solution representation has a significant influence on the obtained results. Selecting an inappropriate solution representation can result in suboptimal performance of the algorithm, which would in turn lead to conclusions that might not be generally applicable. As such, selecting the appropriate solution representation is extremely important, because it eliminates the influence which it could have on the result, and leads to conclusions that are not biased by the underlying solution representation. The results and discussion presented in this paper should provide guidelines for selecting the appropriate solution representation for the unrelated machines environment, depending on the specific requirements placed upon the solution method.

Since the performed experiments denote that the encodings which are based on the permutation representation lead to better results, further research will be directed towards improving those representations. For the PE some more sophisticated heuristics for determining the allocation of jobs to machines will be proposed and tested. For MLE some sophisticated genetic operators will be defined to try and improve its convergence. Finally, the best encodings from this study would also be adapted for solving scheduling problems with additional constraints that were not considered in this paper.

## References


Allahverdi, A., Gupta, J. N., & Aldowaisan, T. (1999). A review of scheduling research involving setup considerations. *Omega*, *27*, 219–239.

Allahverdi, A., Ng, C., Cheng, T., & Kovalyov, M. Y. (2008). A survey of scheduling problems with setup times or costs. *European Journal of Operational Research*, *187*, 985–1032. URL: `http://linkinghub.elsevier.com/retrieve/pii/S0377221706008174`. doi:`10.1016/j.ejor.2006.06.060`.

Baker, K. R., & Trietsch, D. (2013). *Principles of Sequencing and Scheduling*. Wiley.

Balin, S. (2011). Non-identical parallel machine scheduling using genetic algorithm. *Expert Systems with Applications*, *38*, 6814–6821. URL: `http://linkinghub.elsevier.com/retrieve/pii/S0957417410014272`. doi:`10.1016/j.eswa.2010.12.064`.

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, *6*, 154–160. doi:10.1287/ijoc.6.2.154.

Behnamian, J., Zandieh, M., & Fatemi Ghomi, S. (2009). Parallel-machine scheduling problems with sequence-dependent setup times using an ACO, SA and VNS hybrid algorithm. *Expert Systems with Applications*, *36*, 9637–9644. URL: http://linkinghub.elsevier.com/retrieve/pii/S0957417408007252. doi:10.1016/j.eswa.2008.10.007.

Branke, J., Nguyen, S., Pickardt, C. W., & Zhang, M. (2016). Automated Design of Production Scheduling Heuristics: A Review. *IEEE Transactions on Evolutionary Computation*, *20*, 110–124. URL: http://ieeexplore.ieee.org/document/7101236/. doi:10.1109/TEVC.2015.2429314.

Braun, T. D., Siegel, H. J., Beck, N., Bölöni, L. L., Maheswaran, M., Reuther, A. I., Robertson, J. P., Theys, M. D., Yao, B., Hensgen, D., & Freund, R. F. (2001). A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, *61*, 810–837. URL: http://linkinghub.elsevier.com/retrieve/pii/S0743731500917143. doi:10.1006/jpdc.2000.1714.

de C. M. Nogueira, J. P., Arroyo, J. E. C., Villadiego, H. M. M., & Goncalves, L. B. (2014). Hybrid GRASP Heuristics to Solve an Unrelated Parallel Machine Scheduling Problem with Earliness and Tardiness Penalties. *Electronic Notes in Theoretical Computer Science*, *302*, 53–72. URL: http://linkinghub.elsevier.com/retrieve/pii/S1571066114000218. doi:10.1016/j.entcs.2014.01.020.

Carter, A. E., & Ragsdale, C. T. (2006). A new approach to solving the multiple traveling salesperson problem using genetic algorithms. *European Journal of Operational Research*, *175*, 246 – 257. URL: http://www.sciencedirect.com/science/article/pii/S0377221705004236. doi:https://doi.org/10.1016/j.ejor.2005.04.027.

Chang, P.-C., & Chen, S.-H. (2011). Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times. *Applied Soft Computing*, *11*, 1263 – 1274. URL: http://www.sciencedirect.com/science/article/pii/S1568494610000694. doi:https://doi.org/10.1016/j.asoc.2010.03.003.

Chen, B., Potts, C. N., & Woeginger, G. J. (1998). A review of machine scheduling: Complexity, algorithms and approximability. In *Handbook of Combinatorial Optimization* (pp. 1493–1641). Springer US.

Chyu, C.-C., & Chang, W.-S. (2010). A competitive evolution strategy memetic algorithm for unrelated parallel machine scheduling to minimize total weighted tardiness and flow time. In *The 40th International Conference on Computers & Indutrial Engineering* (pp. 1–6).

IEEE. URL: http://ieeexplore.ieee.org/document/5668388/. doi:10.1109/ICCIE.2010.5668388.

Costa, A., Cappadonna, F. A., & Fichera, S. (2013). A hybrid genetic algorithm for job sequencing and worker allocation in parallel unrelated machines with sequence-dependent setup times. *The International Journal of Advanced Manufacturing Technology*, *69*, 2799–2817. URL: http://link.springer.com/10.1007/s00170-013-5221-5. doi:10.1007/s00170-013-5221-5.

Cota, L. P., Haddad, M. N., Souza, M. J. F., & Coelho, V. N. (2014). AIRP: A heuristic algorithm for solving the unrelated parallel machine scheduling problem. In *2014 IEEE Congress on Evolutionary Computation (CEC)* (pp. 1855–1862). IEEE. URL: http://ieeexplore.ieee.org/document/6900245/. doi:10.1109/CEC.2014.6900245.

Eiben, A., & Smith, J. (2015). *Introduction to Evolutionary Computing*. Natural Computing Series. Berlin, Heidelberg: Springer Berlin Heidelberg. URL: http://link.springer.com/10.1007/978-3-662-44874-8. doi:10.1007/978-3-662-44874-8.

Fanjul-Peyro, L., & Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, *207*, 55–69. URL: http://linkinghub.elsevier.com/retrieve/pii/S0377221710002572. doi:10.1016/j.ejor.2010.03.030.

Fanjul-Peyro, L., & Ruiz, R. (2011). Size-reduction heuristics for the unrelated parallel machines scheduling problem. *Computers & Operations Research*, *38*, 301–309. URL: http://linkinghub.elsevier.com/retrieve/pii/S0305054810001188. doi:10.1016/j.cor.2010.05.005.

Fanjul-Peyro, L., & Ruiz, R. (2012). Scheduling unrelated parallel machines with optional machines and jobs selection. *Computers & Operations Research*, *39*, 1745–1753. URL: http://linkinghub.elsevier.com/retrieve/pii/S0305054811003005. doi:10.1016/j.cor.2011.10.012.

Glass, C., Potts, C., & Shade, P. (1994). Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modelling*, *20*, 41–52. URL: https://doi.org/10.1016/0895-7177(94)90205-4. doi:10.1016/0895-7177(94)90205-4.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. (1st ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A. H. G. R. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. *Annals of Discrete Mathematics*, *5*, 287–326. doi:10.1016/S0167-5060(08)70356-X.

Haddad, M. N., Coelho, I. M., Souza, M. J. F., Ochi, L. S., Santos, H. G., & Martins, A. X. (2012). GARP: A New Genetic Algorithm for the Unrelated Parallel Machine Scheduling Problem with Setup Times. In *2012 31st International Conference of the Chilean Computer Science Society* (pp. 152–160). IEEE. URL: http://ieeexplore.ieee.org/document/6694085/. doi:10.1109/SCCC.2012.25.

Hart, E., Ross, P., & Corne, D. (2005). Evolutionary Scheduling: A Review. *Genetic Programming and Evolvable Machines*, *6*, 191–220. URL: http://link.springer.com/10.1007/s10710-005-7580-7. doi:10.1007/s10710-005-7580-7.

Hop, N. V., & Nagarur, N. N. (2004). The scheduling problem of pcbs for multiple non-identical parallel machines. *European Journal of Operational Research*, *158*, 577 – 594. URL: http://www.sciencedirect.com/science/article/pii/S037722170300376X. doi:https://doi.org/10.1016/S0377-2217(03)00376-X.

Kim, C. O., & Shin, H. J. (2003). Scheduling jobs on parallel machines: a restricted tabu search approach. *The International Journal of Advanced Manufacturing Technology*, *22*, 278–287. URL: https://doi.org/10.1007/s00170-002-1472-2. doi:10.1007/s00170-002-1472-2.

Kim, D.-W., Kim, K.-H., Jang, W., & Chen, F. F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer-Integrated Manufacturing*, *18*, 223 – 231. URL: http://www.sciencedirect.com/science/article/pii/S0736584502000133. doi:https://doi.org/10.1016/S0736-5845(02)00013-3. 11th International Conference on Flexible Automation and Intelligent Manufacturing.

Kim, D.-W., Na, D.-G., & Chen, F. F. (2003). Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer-Integrated Manufacturing*, *19*, 173 – 181. URL: http://www.sciencedirect.com/science/article/pii/S0736584502000777. doi:https://doi.org/10.1016/S0736-5845(02)00077-7. 12th International Conference on Flexible Automation and Intelligent Manufacturing.

Lee, J.-H., Yu, J.-M., & Lee, D.-H. (2013). A tabu search algorithm for unrelated parallel machine scheduling with sequence- and machine-dependent setups: minimizing total tardiness. *The International Journal of Advanced Manufacturing Technology*, *69*, 2081–2089. URL: http://link.springer.com/10.1007/s00170-013-5192-6. doi:10.1007/s00170-013-5192-6.

Lenstra, J. K., Shmoys, D. B., & Tardos, E. (1990). Approximation algorithms for scheduling unrelated parallel machines. *Mathematical Programming*, *46*, 259–271. URL: http://link.springer.com/10.1007/BF01585745. doi:10.1007/BF01585745.

Leung, J. Y.-T. (2004). *Handbook of scheduling : algorithms, models, and performance analysis*. Boca Raton, Fla.: Chapman & Hall/CRC.

Lin, C.-W., Lin, Y.-K., & Hsieh, H.-T. (2013). Ant colony optimization for unrelated parallel machine scheduling. *The International Journal of Advanced Manufacturing Technology*, *67*, 35–45. URL: http://link.springer.com/10.1007/s00170-013-4766-7. doi:10.1007/s00170-013-4766-7.

Logendran, R., McDonell, B., & Smucker, B. (2007). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers & Operations Research*, *34*, 3420–3438. URL: http://linkinghub.elsevier.com/retrieve/pii/S0305054806000438. doi:10.1016/j.cor.2006.02.006.

Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., & Freund, R. F. (1999). Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *Journal of Parallel and Distributed Computing*, *59*, 107–131. URL: http://linkinghub.elsevier.com/retrieve/pii/S0743731599915812. doi:10.1006/jpdc.1999.1581.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press.

Morton, T. E., & Pentico, D. W. (1993). *Heuristic Scheduling Systems*. John Wiley And Sons, Inc.

Pinedo, M. L. (2012). *Scheduling: Theory, algorithms, and systems: Fourth edition* volume 9781461423614. Boston, MA: Springer US. URL: http://link.springer.com/10.1007/978-1-4614-2361-4. doi:10.1007/978-1-4614-2361-4. arXiv:arXiv:1011.1669v3.

Raja, K., Arumugam, C., & Selladurai, V. (2008). Non-identical parallel-machine scheduling using genetic algorithm and fuzzy logic approach. *International Journal of Services and Operations Management*, *4*, 72–101. doi:10.1504/IJSOM.2008.015941.

Rocha, P. L., Ravetti, M. G., Mateus, G. R., & Pardalos, P. M. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*, *35*, 1250–1264. doi:10.1016/j.cor.2006.07.015.

Srivastava, B. (1998). An effective heuristic for minimising makespan on unrelated parallel machines. *Journal of the Operational Research Society*, *49*, 886–894. doi:10.1057/palgrave.jors.2600547.

Đurasević, M., & Jakobović, D. (2016). Comparison of solution representations for scheduling in the unrelated machines environment. In *2016 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 1336–1342). IEEE. URL: http://ieeexplore.ieee.org/document/7522347/. doi:10.1109/MIPRO.2016.7522347.

Đurasević, M., & Jakobović, D. (2018). A survey of dispatching rules for the dynamic unrelated machines environment. *Expert Systems with Applications*, . URL: http://www.sciencedirect.com/science/article/pii/S0957417418304159. doi:https://doi.org/10.1016/j.eswa.2018.06.053.

Đurasević, M., & Jakobović, D. (2017a). Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment. *Genetic Programming and Evolvable Machines*, . URL: https://doi.org/10.1007/s10710-017-9302-3. doi:10.1007/s10710-017-9302-3.

Đurasević, M., & Jakobović, D. (2017b). Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment. *Genetic Programming and Evolvable Machines*, . URL: https://doi.org/10.1007/s10710-017-9310-3. doi:10.1007/s10710-017-9310-3.

Đurasević, M., Jakobović, D., & Knežević, K. (2016). Adaptive scheduling on unrelated machines with genetic programming. *Applied Soft Computing*, *48*, 419–430. URL: http://linkinghub.elsevier.com/retrieve/pii/S1568494616303519. doi:10.1016/j.asoc.2016.07.025.

Vallada, E., & Ruiz, R. (2011). A genetic algorithm for the unrelated parallel machine scheduling problem with sequence dependent setup times. *European Journal of Operational Research*, *211*, 612–622. URL: http://linkinghub.elsevier.com/retrieve/pii/S0377221711000142. doi:10.1016/j.ejor.2011.01.011.

Wang, I.-L., Wang, Y.-C., & Chen, C.-W. (2013). Scheduling unrelated parallel machines in semiconductor manufacturing by problem reduction and local search heuristics. *Flexible Services and Manufacturing Journal*, *25*, 343–366. URL: http://link.springer.com/10.1007/s10696-012-9150-7. doi:10.1007/s10696-012-9150-7.

Wotzlaw, A. (2012). *Scheduling Unrelated Parallel Machines: Algorithms, Complexity, and Performance*. AV Akademikerverlag.