Improving genetic algorithm performance by population initialisation with dispatching rules

Ivan Vlašić[1], Marko Đurasević[1,], Domagoj Jakobović[1]

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 is an important process that is present in many real world scenarios where it is essential to obtain the best possible results. The performance and execution time of algorithms that are used for solving scheduling problems are constantly improved. Although metaheuristic methods by themselves already obtain good results, many studies focus on improving their performance. One way of improvement is to generate an initial population consisting of individuals with better quality. For that purpose a variety of methods can be designed. The benefit of scheduling problems is that dispatching rules (DRs), which are simple heuristics that provide good solutions for scheduling problems in a small amount of time, can be used for that purpose. The goal of this paper is to analyse whether the performance of genetic algorithms can be improved by using such simple heuristics for initialising the starting population of the algorithm. For that purpose both manual and different kinds of automatically designed DRs were used to initialise the starting population of a genetic algorithm. In case of the manually designed DRs, all existing DRs for the unrelated machines environment were used, whereas the automatically designed DRs were generated by using genetic programming. The obtained results clearly demonstrate that using populations initialised by DRs leads to a significantly better performance of the genetic algorithm, especially when using automatically designed DRs. Furthermore, it is also evident that such a population initialisation strategy also improves the convergence speed of the algorithm, since it allows it to obtain significantly better results in the same amount of time. Additionally, the DRs have almost no influence on the execution speed of the genetic algorithm since they construct the schedule in time which is negligible when compared to the execution of the genetic algorithm. Based on the obtained results it can be concluded that initialising individuals by using DRs significantly improves both the convergence and performance of genetic algorithm, without the need of having to manually design new complicated initialisation procedures and without increasing the execution time of the genetic algorithm.

*Corresponding author

Keywords: Scheduling, Unrelated machines environment, Genetic algorithms, Dispatching rules, Population initialisation

---

Symbols

C_j	completion time of job j
d_j	due date of job j
i	index of machine
j	index of job
m	number of machines
n	number of jobs
p_{ij}	processing time of job j on machine i
r_j	release time of j
T_j	tardiness of job j
Twt	total weighted tardiness of a schedule
w_j	weight of job j

1. Introduction

Scheduling is defined as an optimisation problem in which it is required to determine the allocation of a certain number of jobs to a limited number of machines, so that some user defined criteria are optimised (Pinedo, 2012). The interest for solving scheduling problems is quite high since they appear in many real world situations like airplane scheduling (Cheng et al., 1999b; Hansen, 2004), scheduling in manufacturing and fabrication (Dimopoulos & Zalzala, 2000; Pfund et al., 2006; Chiang et al., 2008; Kofler et al., 2009), scheduling in cloud environments (Zhan et al., 2015; Singh & Chana, 2016), staff scheduling (Ernst et al., 2004), multiprocessor scheduling (Hou et al., 1994), or scheduling patients in hospitals (Petrovic & Castro, 2011). An additional reason why these problems are heavily researched is because most of them are NP-hard. Therefore, it is not possible to develop an algorithm that could obtain optimal solutions for such problems in a reasonable amount of time. As a consequence, scheduling problems are usually solved by a variety of heuristic algorithms. With these heuristics it is possible to rapidly obtain good solutions for different scheduling problems. The solution quality and the time required to construct it will depend on the type of heuristic which is used. Based on the way how the heuristics solve scheduling problems, they are usually divided into *improvement* and *constructive* heuristics.

Constructive heuristics start with an empty solution, after which they iteratively construct the entire schedule. Therefore, constructive heuristics do

not search the entire space of solutions, but rather use a certain kind of strategy to determine how the schedule should be constructed. As a consequence, constructive heuristics can create schedules in a quite small amount of time. Furthermore, since they iteratively build solutions, they do not require that all information about the problem is available at the start, but rather the information can become available as the system is executed. As a result, they can be used for solving scheduling problems under dynamic conditions, where the schedule is constructed simultaneously with the execution of the system and not all information about the problem is available up front. Since they do not search the entire solution space of the problem, but rather use a greedy strategy to construct the solution, they mostly achieve inferior results when compared to improvement heuristics. Constructive heuristics most often appear in the form of *dispatching rules* (DRs) (Đurasević & Jakobović, 2018).

Improvement heuristics start with an initial solution or set of solutions for a specific scheduling problem, and then iteratively try to improve the solutions by using various operations. This means that they perform a search over the solution space of the problem to find the best possible solution. Because of that they are likely to obtain good solutions for different kinds of scheduling problems. Although they can obtain quite good solutions, it is necessary to provide them with enough computational time to reach them. Thus, improvement heuristics usually require a substantially larger amount of time to obtain good solutions than constructive heuristics, but the solutions they obtain are in most cases significantly better. Since these methods start with a concrete solution, they can mostly be applied only for solving scheduling problems under static conditions, in which all the information about the system is available beforehand. There is also a wide range of metaheuristic methods that can be applied, such as genetic algorithms (**?**), particle swarm optimisation (Kennedy & Eberhart, 1995), ant colony optimisation (Colorni et al., 1991), but also many other nature inspired algorithms like the bat algorithm (Gandomi & Yang, 2014), harmony search (Geem et al., 2001), rain-fall optimisation(Kaboli et al., 2017b), and many others (Boussaïd et al., 2013; Gogna & Tayal, 2013). Metaheuristic methods were applied for solving various kinds of optimisation problems like vehicle routing problems (Li et al., 2015), design of cryptographic functions (Picek et al., 2016; Picek et al., 2016), solving the economic dispatch problem (Modiri-Delshad et al., 2016; Kaboli & Alqallaf, 2019), electric power consumption forecasting (Kaboli et al., 2016b,a, 2017a), as well as for many other areas (Fitzgerald et al., 2015; Krawiec & Pawlak, 2015; Soler-Dominguez et al., 2017; Sebtahmadi et al., 2018; Mohamad Izdin Hlal et al., 2019) Different metaheuristic methods have also been applied for solving various scheduling problems in the literature (Cheng et al., 1996; Wang et al., 1997; Cheng et al., 1999a; Zhou et al., 2001; Ishibuchi et al., 2003; Hart et al., 2005; Gao et al., 2007; Vallada & Ruiz, 2011; Lin et al., 2013; Lee et al., 2013).

The performance of improvement heuristics depends on many different parameters, and it is required to perform an initial parameter optimisation step to improve the odds of obtaining good solutions. Aside from the algorithm parameters, the initial population of solutions has a significant influence on the

quality of the results, but also on the convergence speed of the algorithms. In most cases the initial populations which are used by the improvement heuristics are constructed randomly. As a consequence, the initial population usually consists out of solutions of poor quality. Therefore, the improvement heuristics have to invest time to construct solutions with good characteristics. For that reason, several papers focused on analysing the influence of different population initialisation mechanisms to enhance the performance of improvement heuristics (Rahnamayan et al., 2007a; Diaz-Gomez & Hougen, 2007; Rahnamayan et al., 2007b; Kazimipour et al., 2013, 2014).

Unfortunately, for solving scheduling problems there is not a substantial amount of research which is focused on initialisation techniques of improvement heuristics. Burke et al. (1998) analysed the influence of several heuristic initialisation strategies on the quality and diversity of the initial population. Based on the results obtained in the paper, the authors concluded that using the proposed initialisation strategies evolutionary algorithms can perform much better. However, the methods proposed in that study are applicable for the timetabling and other related problems, which means that the methods used for population initialisation can not be applied to other kinds of scheduling problems without significant changes. Yang et al. (2009) propose a novel initialisation method which is based on two sub-methods, the global and local selection. These methods were used to initialise the population of a genetic algorithm when solving the flexible job shop problem. The proposed initialisation method works similarly as many dispatching rules since it tries to allocate the selected job and its operations to the machines that could complete those operations the soonest. The obtained results show that the proposed initialisation method can improve the performance and computational time of the algorithm. The drawback of this proposed method is that when constructing the initial population it randomly selects and orders the jobs, rather than using some more sophisticated ordering procedure which would lead to even better initial populations. Vigneswari & Mohamed (2014) apply the artificial bee colony algorithm with three initialisation procedures for solving the grid scheduling problem. The results show that by using the initialisation strategies the genetic algorithm achieves a better performance than by using random initialisation. However, the initialisation techniques proposed in this study are algorithm specific, which means that they are applicable only for the considered artificial bee algorithm and do not use any information about the scheduling problem that was considered. Sarathambekai & Umamaheswari (2017) use a discrete particle swarm optimisation algorithm which is enhanced with the opposition-based technique for generating the initial solutions for scheduling tasks in multiprocessor environments. The proposed initialisation method allocates the jobs on the machines based on their load, which has shown to perform well for the makespan and flowtime related criteria. However, this method also focuses on just the allocation aspect of the scheduling problem, and not on the order in which the jobs will be executed.

In most of the previously outlined studies completely new initialisation techniques were developed. However, designing new initialisation techniques is a difficult trial and error task. This is due to the reason that a great variety of

scheduling problems exist, and thus appropriate initialisation procedures would need to be designed for all the different variants. Therefore, many studies also based their initialisation techniques on existing DRs, since this omits the need to design new initialisation procedures. Xhafa & Abraham (2008) outline that DRs can be used to generate several good solutions in the initial population which could then accelerate the search. However, the authors do not provide any investigation of the effect of using DRs for initialising the population of the genetic algorithm. Xiong et al. (2012) used several DRs in the initialisation procedure of the proposed multi-objective evolutionary algorithm to improve the quality of the initial population. This study only compared the proposed algorithm which included such population initialisation strategies with some other algorithms, and thus the benefit and influence of the population initialisation with DRs was is not clear from the study, since the proposed algorithm also includes other improvements like local search. Han et al. (2016) use the MME heuristic, which represents a combination of two popular DRs, for generating the initial population for the fruit fly optimisation algorithm. This study also did not analyse the effect that the proposed initialisation method had on the algorithm, rather it just compared the modified algorithm with other algorithms. Unlike in the previously outlined studies where manually designed DRs were used, Kuczapski et al. (2010) evolved new DRs which were used for initialising the starting population. The rules were represented as composite DRs, which represents a weighted sum of several popular DRs. Then by using a genetic algorithm the weights for the composite DR are optimised. The experimental results demonstrate that the proposed population initialisation strategy leads to better results than by using only randomly generated solutions in certain cases. Unfortunately the experimental results and discussion provided in the paper were not very extensive. Furthermore, the way in which the DRs were designed, by using a weighted sum, has in similar research proven to be inferior to other methods of automatically designing DRs which are based on genetic programming and similar methods (Branke et al., 2015).

As can be seen from the outlined literature review, the initial population for solving scheduling problems can either be initialised by using specifically designed initialisation strategies or by using existing simple heuristic procedures, most notably DRs. Designing initialisation strategies for the various problem variants and optimisation criteria would be quite time consuming. Thus even though such initialisation procedures might be effective, the need to always design new strategies represents a significant drawback. This would be especially problematic for problems that are more complex and not standard. As a consequence, initialising the population by using DRs seems to be a more favourable approach. The reason for this is that a lot of different DRs exist for various kinds of scheduling problems which could be reused for initialising the population of genetic algorithms. In previous studies the initial population still consisted mostly out of randomly designed solutions, while only several solutions were generated by using DRs. Most of the studies used population initialisation as an additional part of their newly designed algorithms, and did not provide a detailed analysis on how that part actually influences the per-

formance. Furthermore, most studies used manually designed DRs to generate the initial population. Therefore, one could also argue that there are scheduling problems for which no adequate DRs exist, which could be used to initialise the population of the genetic algorithm. However, in recent years a great deal of research was done in the area of automatic DR design by using genetic programming (GP) (Branke et al., 2016; Nguyen et al., 2017). These DRs obtain mostly better results than manually designed DRs, which could also make them more suitable for initial population generation. Furthermore, GP can be used to design DRs for any kind of scheduling problem which eliminates the need of manual design of novel population initialisation methods. In the end, DRs do provide more flexibility when generating the initial population for metaheuristic methods. However, the existing research still leaves many questions open regarding to this topic, which is mostly due to the fact that population initialisation was used with other methods (like local search) to improve the performance and no detailed analyses were performed to asses the influence of population initialisation via DRs. Therefore the following questions still remain open in the currently available studies:

1. How much influence does initialisation with DRs have on the performance and the convergence of genetic algorithms?
2. Are automatically designed DRs better for initialising the starting population than the manually designed ones?
3. How does the ratio between randomly initialised and DR initialised individuals in the population influence the performance?
4. What influence does initialisation with DRs have on the diversity of the population?

The objective of this paper is to investigate how the performance of a genetic algorithm can be improved by constructing an initial population with a better quality. The main goal is to clearly denote and analyse the influence of the DR initialisation strategies on the performance and convergence speed of the genetic algorithm, depending on whether manually or automatically designed DRs are used to initialise the population. Another focus is to analyse the influence of the proposed initialisation methods on the population diversity and consequentially the algorithm performance. The genetic algorithm was selected since it is one of the most commonly used improvement heuristics which is used in the literature to solve scheduling problems Hart et al. (2005). However, the tested initialisation procedures can be used to initialise the populations of any other population based methods. This paper will specifically focus on the construction of the initial population by using DRs. Since DRs can create schedules of a relatively good quality in a small amount of time, they would not increase the execution time of the genetic algorithm substantially, but would lead to the construction of a better initial population. By using DRs it is not necessary to define any complex initialisation strategies, due to the reason that these DRs can be designed automatically which decrease the effort required for developing them. The paper will also analyse the difference in the performance of the genetic algorithm when manually or automatically designed DRs are

used to create the initial population. The contributions of this paper can be summarized as:

1. Application of manually and automatically designed DRs for population initialisation of GAs
2. Performance comparison of initialisation strategies using different DR groups and population sizes
3. Convergence analysis of the algorithm when using different DRs for initialisation
4. Diversity analysis of the populations constructed by different initialisation methods

The rest of the paper is organised as follows. Section 2 provides an introduction into the topics of scheduling problems, genetic algorithms, and DRs. The population initialisation strategies that will be analysed in this paper are described in Section 3. The experimental set-up is explained in Section 4. The obtained results are presented in Section 5, while Section 6 provides a short discussion based on the obtained results. Finally, Section 7 provides a short conclusion and outlines possible future research directions in this topic.

2. Background

2.1. Scheduling

Since scheduling problems are present in many real world situations, they have attracted a lot of attention from researchers. In order to model the various situations that can appear in the real world, several machine models have been proposed (Pinedo, 2012). In this paper the *unrelated machines* scheduling environment will be used. This environment specifies that each job needs to be assigned on one of the available machines for execution. However, in this environment each machine executes the jobs with a different speed. Therefore, no relations between the different machines can be deduced. For example, one machine can execute one job faster than another machine, while all other jobs can be executed faster on the second machine. This property makes solving problems in this environment quite challenging.

When specifying the scheduling problem, the number of machines in the problem is usually denoted with m, while the number of jobs is denoted as n. The index i is used to denote a certain machine, while the index j is used to denote a specific job. For each job several properties have to be specified. The most important property is the processing time of job j on machine i, which is denoted as p_{ij}. This is the only property that has to be specified, since all others depend on the type of the scheduling problem and criterion which is optimised. If the jobs are not available at the start of the system, but rather they arrive during the execution of the system, then a release time for is specified for each job. This property is denoted as r_j and represents the time when the job is released into the system. In certain cases each job can also have a due date, which is denoted as d_j. The due date defines the point in time until which a

job should be executed. If the job is not finished until then, a certain cost or penalty is invoked. This penalty becomes larger as the time after which the job is executed after its due date increases. This property is only specified and used if due date related criteria have to be optimised. Finally, all the jobs are usually not equally important. For example, some jobs need to be executed as soon as possible, and if they are not executed on time a larger penalty will be invoked than in the case if some other jobs would be late. In order to denote that some jobs are more important than others a weight, denoted as w_j, is associated with each job. A larger weight denotes that the job is more important and that it should be scheduled sooner.

By using the aforementioned properties a solution can be constructed and evaluated by using different scheduling criteria (Allahverdi et al., 1999, 2008; Durasević & Jakobović, 2018). The scheduling criterion which will be optimised depends largely on the user requirements. In this paper the time which the jobs finish after their due date will be minimised. In order to do that, a measure which calculates the time that the job spent executing after its due date is defined. This measure is called *tardiness* and is calculated as $T_j = \max(C_j - d_j, 0)$, where T_j represents the tardiness of job j, and C_j represents the completion time of job j. Based on this measure the *total weighted tardiness* is calculated as $Twt = \sum_j w_j T_j$. As it can be seen from the definition, this criterion also uses the job weights to give more importance to certain jobs, which means that more important jobs will have a larger influence on this criterion if they are tardy. This criterion specifies the total tardiness of all jobs in the schedule, and needs to be minimised.

2.2. Genetic algorithm

Genetic algorithm (GA) is a popular metaheuristic optimisation method which is used to solve various kinds of problems, including different kinds of scheduling problems. The first step which needs to be performed when applying a GA for solving a problem is to select the solution representation. Although various solution representations have been proposed for solving scheduling problems (Costa et al., 2013; Đurasević & Jakobović, 2016; Bean, 1994; Behnamian et al., 2009; Balin, 2011), in this paper the solutions will be represented by using the *machine list encoding* (MLE) (Vallada & Ruiz, 2011). The reason why this solution representation was selected is not only due to its simplicity, but also because in some preliminary experiments it obtained the best results out of several tested solution representations. Figure 1 shows a solution represented by using the MLE for a scheduling problem consisting of three machines and nine jobs. In this encoding for each machine there is a list of jobs that need to be executed on the specific machine. The jobs are executed on the machine in the order in which they appear in the list. In this example, jobs 3 and 7 would be executed on machine 0, jobs 5, 1, 2, and 0 would be executed on machine 1, while jobs 6, 4, and 8 would be executed on machine 2.

Aside from specifying the solution representation, it is additionally required to select the genetic operators which will be used. For the crossover operator the *point crossover* will be used. An example of this crossover is shown in Figure

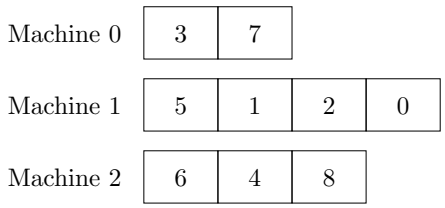


Figure 1: A solution represented by the machine list encoding

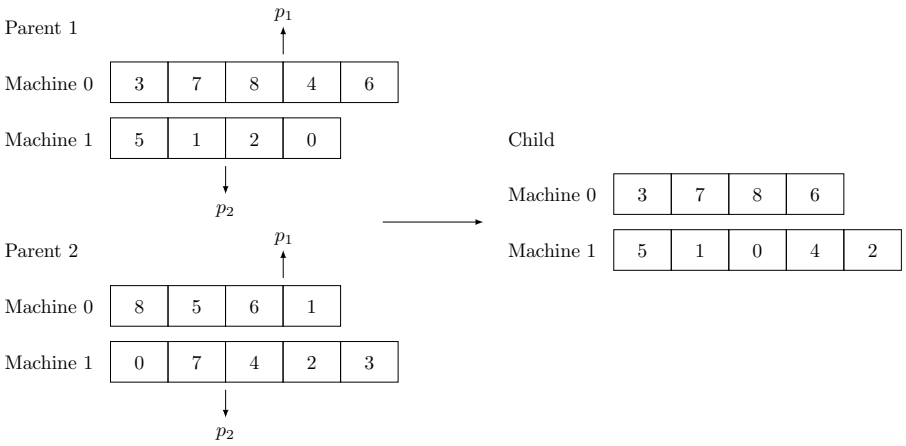
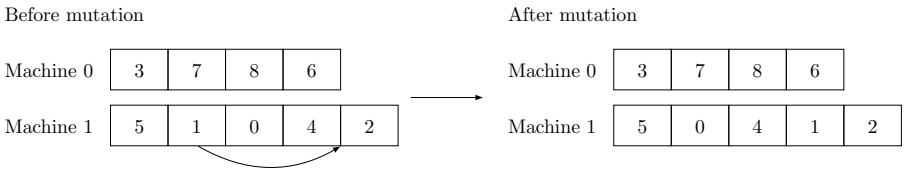
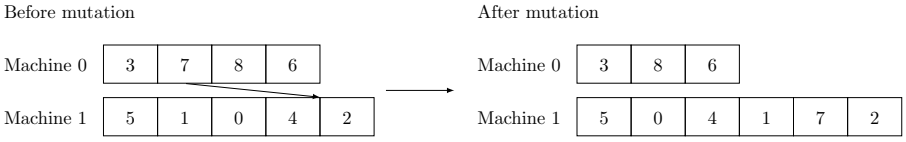


Figure 2: Point crossover used for MLE

2. In this crossover for each machine permutation list a random crossover point is selected. The child individual is constructed in a way that all jobs which appear before the crossover point in the first parent are directly copied into the child individual. To ensure that the lists in the child individual represent valid permutations, it is not possible to copy all the jobs after the crossover point from the second parent. Instead, all jobs in a permutation list of the second parent will be traversed, and for each job it will be checked whether it already appears in any of the machine lists of the child individual. If it does, it will be skipped, otherwise it will be placed on the corresponding machine in the child individual. In this way it is ensured that the child individual represents a valid solution. In the example in Figure 2 the crossover point for the first machine was selected so that the first three jobs are taken from the first parent, while for the second machine only the first two jobs would be taken from the first parent. After that, the list for the first machine in the second parent is traversed, during which job 6 is put into the permutation list of the child, while jobs 5 and 1 are skipped since they are already present in the list of the second machine. The same procedure is also used to determine which jobs will be placed on the second machine in the child individual.



(a) Insertion on the same machine



(b) Insertion on a different machine

Figure 3: Insert mutation used for MLE

For the mutation operator the *insert* mutation will be used. In this mutation a random job is selected and a new position is randomly chosen for it. Figure 3 shows an example of the insert mutation. In this operator two different cases can occur. The first case is that the job is placed on the same machine, but on a different position, which is denoted in Figure 3a. In the second case, the job is scheduled on a completely different machine and placed on a random position, which is denoted in Figure 3b. Since the position on the second machine is randomly selected, it can be either different or even the same as the position on the initial machine. Since the mutation can not result in invalid permutations there is no need to have additional constraints in the mutation operator.

2.3. Dispatching rules

As denoted in the introduction, DRs are simple heuristics that create the schedule incrementally. This is done in a way that at each *decision point* they determine which job should be scheduled next on which machine. A decision point can be defined as a moment in time when a job is released and there are available machines in the system, or a machine becomes available and there are unscheduled jobs in the system. The selection of a job is usually made by calculating the priority for each job by using certain job and system parameters. For example the priority of job j can be calculated by the function

$$f(j) = \frac{1}{r_j},$$

which would simply calculate the priority based on the release times of the jobs. The job with the largest priority is then selected and scheduled on the appropriate machine. For the aforementioned priority function the job which was released the earliest would have the highest priority and would be scheduled first. The DR then moves to the next decision point, and again determines which jobs should be scheduled. The procedure is repeated until all jobs are scheduled.

Naturally, there are DRs of different complexities, ranging from those simple which make the decision based only on one or two job and system parameters, to those more complex which define more sophisticated scheduling procedures Maheswaran et al. (1999); Braun et al. (2001); Đurasević & Jakobović (2018). Furthermore, the definition of DRs also depends on the criterion which need to be optimised, since different system and job properties will be used when optimising different scheduling criteria.

One additional property which needs to be specified for DRs is which jobs are considered at each decision point. The type of rules which consider only those jobs which were released until the decision point are designed for solving *dynamic scheduling problems*. These DRs assume that the information about the unreleased jobs is not available, and thus they do not use it. This makes the rules simpler and faster, but also more myopic if static information is available, since by using that information they could perform better decisions and obtain better results. Most of the DRs proposed in the literature are of this type. However, DRs can be extended to also consider jobs which are not yet released at the current decision point (Branke & Pickardt, 2011). These rules also calculate priorities for jobs that are yet unreleased, and if one such job has the highest priority it will be scheduled instead. Such rules will have a better overview on the problem since they will have the possibility to introduce idle times in the schedule if they determine that a more important job will soon be released into the system. As a result, such rules will achieve better solutions than DRs designed for dynamic scheduling problems.

3. Population initialisation by dispatching rules

As denoted in the introduction, the initial population of the GA will be initialised by using four different DR initialisation strategies. Each of the initialisation strategies mentioned in this section has certain benefits and drawbacks, which will be discussed in detail in the later part of the paper.

The first initialisation strategy will use a set that consists out of 26 manually designed DRs for solving the unrelated machines scheduling problem, described in (Đurasević & Jakobović, 2018). The set includes many popular and commonly used DRs like ATC (Lee et al., 1997), min-min (Maheswaran et al., 1999), and sufferage (Maheswaran et al., 1999). Unfortunately the set of manually designed DRs is quite limited. Since designing new DRs manually is a difficult and time consuming process, it is not expected that the number of such DRs will significantly increase in the future. Thus the number of solutions that can be generated by them is also limited. Furthermore, not all of the included DRs are designed to optimise the Twt criterion. Therefore, not all the solutions in the initial population will have a good quality for that criterion. All of the DRs in this set consider only the released jobs in each of the decision points, which means that they are designed to deal with problems under dynamic conditions. The initialisation strategy which will initialise the population by using the solutions generated by manually designed DRs will be denoted as DEX.

The second initialisation strategy uses a set of DRs which consists out of DRs automatically designed by using GP. The DRs were designed as described in a previous study which dealt with the automatic generation of DRs for the unrelated machines environment (Đurasević et al., 2016). The DRs generated for this set were also designed so that they consider only the jobs which were released until the current decision point. All the DRs in this set were generated for optimising the Twt criterion, which means that all of them should obtain good results for the given criterion. The benefit of using DRs generated by GP is that a large number of such DRs can be generated without significant effort. Furthermore, in most cases these automatically generated DRs obtain better results than manually designed DRs, which makes them more favourable. The initialisation strategy that will use this set of DRs will be denoted as DGP.

The problem with the previous two DR sets is that the DRs in both of them are designed for dynamic scheduling, whereas the GA is used for solving static problems. This means that the DRs in the previous two sets will construct solutions by using only partial information about the problem. This will result in solutions that are worse than those that could be obtained if all the information about the problem had been used. Therefore, the final set of DRs consists out of rules which are designed for solving static scheduling problems. This should result in a better starting population since the DRs will use all the information about the scheduling problem (similarly as the GA) and will thus be able to construct better schedules. The DRs in this set will use look-ahead, which will allow them to take into account all the jobs that have to be scheduled. All the DRs in this set were also generated by using GP. The initialisation strategy that will use this set of DRs will be denoted as SGP in the results.

Aside from the previous three initialisation strategies, an additional strategy will be used which initialises the population by using the union of the results obtained by the three above mention initialisation strategies. The idea behind this initialisation strategy is to test whether a combination of the three different DR types could be beneficial to the GA, since it would lead to a more diverse initial population. The strategy that will initialise the individuals as a union of all the three aforementioned strategies will be denoted as CMB in the experiments.

4. Experimental setup

The aforementioned initialisation strategies will be tested for solving the scheduling problem which can be denoted as $Rm|r_j|Twt$ (Pinedo, 2012). This notation denotes that the unrelated machines scheduling problem subject to job release times is solved, and that the total weighted tardiness criterion is optimised. To test the influence of the different initialisation strategies, a set of problem instances will be used to measure the performance of each strategy. The problem set which was used to test the performance of the initialisation strategies consists out of 60 randomly generated problem instances. The number of jobs n can range from 12 to 100, while the number of machines m can range from 3 to 10. Furthermore, the job due dates are also generated with different parameters to simulate problems in which more jobs will be late, or

problems in which all jobs can complete prior to their due dates. Since the problem set consists out of 60 problem instances, the GA is executed for each problem instance independently and the total fitness on the entire problem set is calculated as the sum of the Twt values for each of the individual problem instances. More about the problem instance generation procedure can be found in (Đurasević et al., 2016).

For solving the aforementioned scheduling problem a steady state tournament GA will be used. The parameters of the GA were previously optimised to obtain the best possible results. The algorithm will use a population size of 30 individuals when no initialisation strategy is used, or when the population size is not specified. For the mutation the probability of 0.9 will be used. The tournament will consist out of three individuals, where the two better ones are recombined and the worst one is eliminated. The algorithm is set to terminate after one million function evaluations. In this way the GA will perform the same amount of work for all initialisation strategies and different population sizes.

To test the performance of the different population initialisation strategies, each of them will first be used to generate the initial population without any additional randomly generated individuals. In this case, the GA will use different population sizes, which adhere to the number of DRs in each strategy that were used to generate the initial schedules. Since for the manually designed DRs only 26 rules were found in the literature, the GA will use a population consisting out of 26 individuals. For the automatically generated DRs designed for dynamic scheduling conditions 50 good DRs were obtained, the population size will be fixed to that number. Finally, 30 good DRs were automatically generated for static scheduling conditions, therefore the GA will use a population size of 30 individuals for this initialisation strategy. When all three initialisation strategies are combined, then the population size for the GA will be equal to the sum of the population sizes for the individual initialisation strategies, which totals to 106 individuals. Aside from using populations which consists only out of individuals generated by DRs, the GA will also be tested in situations where larger population sizes are used. In these cases the individuals which were not initialised by the selected initialisation strategy are simply generated randomly. In these tests the GA will use population sizes of 150, 200, and 500 individuals.

Each experiment in this paper is executed 30 times to ensure that statistically significant results were obtained. To test whether the results obtained by the various initialisation strategies and population sizes are significantly different from one another, the Mann-Whitney statistical test will be used. To asses that two results are significantly different from each other the p value needs to be less than 0.05.

5. Results

5.1. Performance comparison of the initialisation strategies

In this section the results for the tested initialisation strategies will be presented. Table 2 represents the results obtained for the different initialisation

strategies, as well as for the various tested population sizes. The entry denoted with "-" for the population size represents that the population sizes for each initialisation strategy are different. This is due to the fact that each initialisation strategy will initialise the population only with solutions generated by DRs, which causes that for each strategy the GA will have a different population size. When using the random (RND) initialisation, the GA will use a population size of 30 individuals. For each population size and initialisation strategy four metrics are calculated: the minimum, median, maximum, and total minimum (Tmin). The total minimum metric represents the fitness value which is calculated by summing the best fitness value (obtained by any of the 30 independent executions of the GA) for each problem instance. Therefore, this measure represents the best possible results that were obtained by the GA across all 30 executions, and not during only one GA execution. The cells which are greyed out denote the best results for each of the metrics which are denoted in the table. The results are additionally denoted in Figure 4 by using box plots.

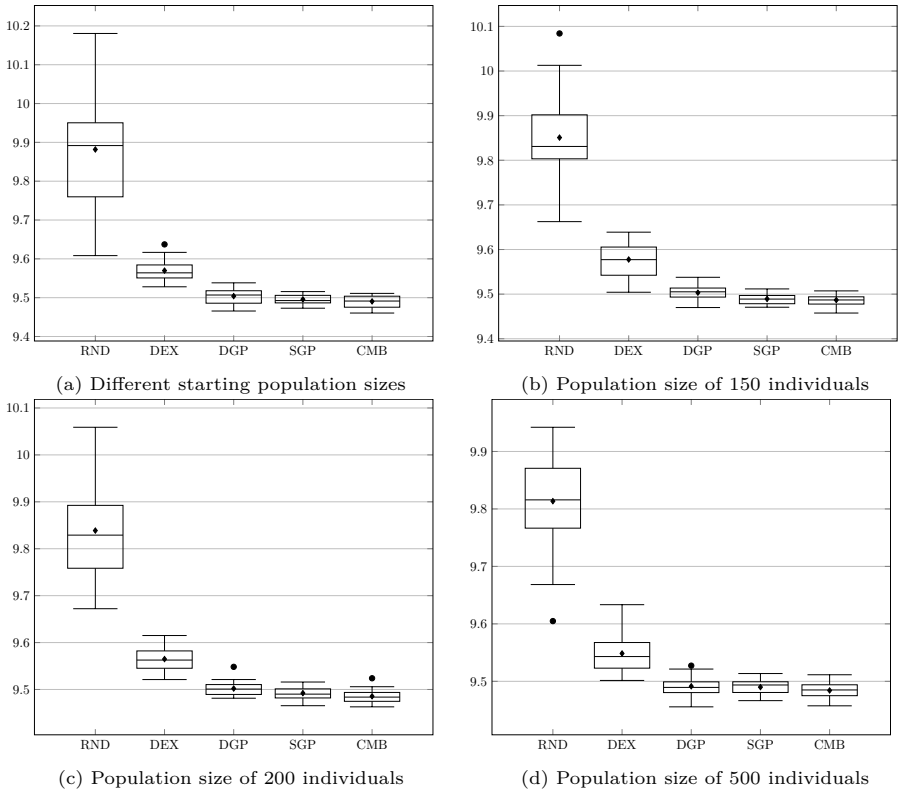
The first thing that can be seen from the results is that the random initialisation strategy obtains the worst results among all the initialisation strategies. This is especially evident from the fact that even the best results obtained in the 30 executions are worse than the worst result obtained in the 30 runs by the other initialisation strategies. This can best be seen for population sizes of 150, 200, and 500 individuals. Furthermore, the results obtained by the RND initialisation strategy are the most dispersed among all the initialisation strategies. This means that to be sure that good results are really obtained for a scheduling problem the GA needs to be executed several times to ensure that it did not get stuck in a local optima. The results of the statistical tests performed between the RND initialisation strategy and the other strategies prove that the RND initialisation strategy obtains significantly worse results. Regarding the different population sizes used by the GA there is almost no significant difference between the results obtained by each of them. Only the GA with the population size of 500 individuals obtained a significantly better result over the GA with the population size of 30 individuals. However, the p value was quite close to 0.05 and thus the significance of this result is not quite strong.

The DEX initialisation strategy obtained much better results than the RND strategy, usually achieving results that better by around 3.4%. However, its performance still falls behind the other initialisation strategies. The figures show that most of the results achieved by the GA with this strategy are worse than most of the results obtained by the remaining three strategies, although the differences are less than 1%. The statistical tests prove that when compared to the three remaining population initialisation strategies, this one achieves significantly worse results. When different population sizes are tried out with this initialisation strategy it can be seen that there are only slight differences in the obtained results. However, when using a population of 500 individuals the GA achieved results which were significantly better than when using any of the smaller population sizes. Therefore, this strategy seems to work best when used with a large number of additionally randomly generated individuals. Furthermore, this initialisation strategy reduces he range of the obtained results

Table 2: Results obtained by testing different initialisation strategies and population sizes

Population size		Initialisation strategies				
		CMB	DEX	DGP	RND	SGP
~	Tmin	9.439	9.499	9.445	9.485	9.451
	Min	9.46	9.528	9.466	9.608	9.473
	Med	9.492	9.566	9.509	9.9	9.493
	Max	9.511	9.637	9.538	10.18	9.516
150	Tmin	9.435	9.462	9.441	9.475	9.456
	Min	9.458	9.504	9.47	9.662	9.471
	Med	9.488	9.579	9.506	9.834	9.491
	Max	9.507	9.639	9.538	10.084	9.512
200	Tmin	9.431	9.472	9.439	9.475	9.453
	Min	9.463	9.521	9.481	9.672	9.465
	Med	9.485	9.563	9.502	9.831	9.491
	Max	9.524	9.615	9.548	10.059	9.516
500	Tmin	9.43	9.45	9.438	9.492	9.453
	Min	9.457	9.502	9.456	9.605	9.466
	Med	9.485	9.545	9.49	9.821	9.494
	Max	9.511	9.633	9.527	9.942	9.514

by a factor of 5 when compared to the RND strategy, which demonstrates just how much the dissipation of the results can be reduced by initialising the population.



(a) Different starting population sizes

(b) Population size of 150 individuals

(c) Population size of 200 individuals

(d) Population size of 500 individuals

Figure 4: Comparison of the results obtained by various population sizes

With the DGP initialisation strategy the GA obtains better results than by initialising the population either randomly or by using manually designed DRs. The improvements over the RND strategy equal to around 4% on the average. By using this initialisation strategy the GA obtains results which are only slightly dispersed. This is a benefit of this strategy since it means that upon several executions of the GA it should obtain similar results. The results denote that this strategy still achieves slightly worse results than the remaining two initialisation strategies, especially for the smaller population sizes. This is also backed up by the statistical tests which show that the results obtained by the GA with DGP are significantly worse then when using either SGP or CMB. For the population size of 500 individuals there is no significant difference between DGP and the remaining two initialisation strategies. Still, it can be concluded that the GA with DGP is inferior than the GA which would use one of the other two initialisation strategies, although the differences between the results on the average are not larger than 0.3%. It seems that this initialisation

strategy also benefits from using a larger population which is additionally filled with randomly generated individuals. The statistical test also backs up this conclusion, since the results obtained when using the population size of 500 are significantly better than the results obtained by the GA for any of the three smaller population sizes. On the other hand, there is no statistically significant difference between the results obtained for the other population sizes.

By using the SGP initialisation strategy the GA achieves the second best results. The improvements over the RND strategy remain at around 4%. Only the the CMB strategy achieves consistently a better median value than the SGP strategy. However, the statistical tests show that there is no significant difference between these two initialisation strategies for all population sizes except for the population size of 500 individuals for which the CMB strategy achieved significantly better results. Even in this case the p value was quite large, which means that the significance of the result is not strong. The box plots also denote that the results obtained by this strategy are also not very dispersed, meaning that the algorithm behaves very stable. Even more, with this initialisation strategy the GA obtains the least dispersed results among all the strategies. Compared to the RND strategy this one achieves a range of solutions which is 13 times smaller. Additionally, it is evident that using additional randomly generated individuals with this initialisation strategy does not lead large differences in the results. The statistical tests back up this conclusion since there is no significant difference between the results obtained for for the different population sizes. Therefore, unlike the previous initialisation strategies, this one does not benefit from using larger population sizes.

Finally, with the CMB initialisation strategy the GA obtains the best possible results, which is also evident from the table, since for most metrics this strategy obtains the best values. The improvements over the RND strategy remain at around 4%. As outlined in the previous paragraphs this strategy obtains significantly better results than all other strategies, except for the SGP strategy with which it achieves mostly the same results. Furthermore, this initialisation strategy also does not benefit greatly from increasing the population size by adding randomly generated individuals, since the statistical tests show that for all the tested population sizes the GA obtains results between which there is no significant difference.

Another interesting thing, which can be observed from the results, is that the GA obtains a similar value for the Tmin metric for all initialisation strategies and population sizes. In all experiments the Tmin value ranged between 9.43 and 9.5. Thus, the GA shows that regardless of the initialisation strategy it can obtain results of the same quality if it executed several times and the best results are collected for each problem instance. In that regard it could even seem that there is actually no benefit behind using other initialisation strategies. However, it should be noted that when using better initialisation strategies like SGP and CMB, the median value obtained on the experiments is much closer to the Tmin value, than when using the random initialisation strategy. For example, for the CMB initialisation strategy the GA obtained median values which are only by around 0.5% worse than those of the Tmin values. This

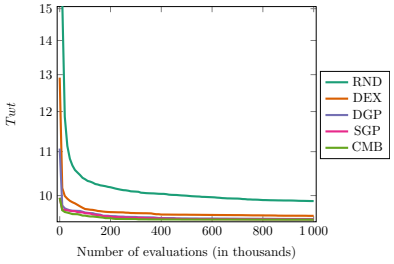
means that with this initialisation strategy the GA is able to converge closely to the best obtained results in almost each of its executions. Therefore, to obtain really good solutions it is not even required to execute the GA several times, but rather good solutions can be obtained even with one GA execution. This further demonstrates the superiority of the GA when using DRs to initialise the starting population.

5.2. Fitness dynamics during the GA execution

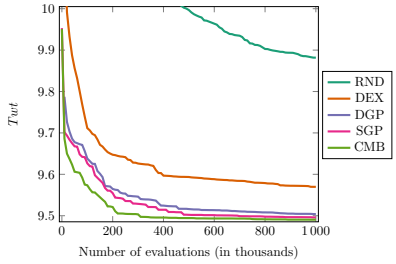
Besides analysing the total results after the execution of the GA, it is also interesting to observe how the fitness changes during the execution of the GA. Figure 5 represents the average fitness of the best individual on the 30 executions. Each figure denotes how the fitness changes with the number of evaluations which were performed by the algorithm. Since it can be difficult to see differences between some initialisation strategies, additional zoomed in figures were also included. The figures show how with the different initialisation strategies the GA starts with populations of vastly different qualities. As expected, the worst initial population quality is obtained by random initialisation, which usually generates a population that has an average Twt value between 500 and 700. On the other hand, all the other population initialisation strategies start with with a population that has an average Twt value between 9.9 and 13. Thus the other initialisation strategies start with a population that is better by more than one order of magnitude. Therefore, when using the random initialisation strategy the GA has to invest a lot of time to reach good solutions, whereas the other initialisation strategies already start with good solutions which the GA can fine tune during the evolution process.

From the graphs it is evident that at the start of the evolution process the GA heavily explores the search space and thus quickly improves the quality of its best solution. Thus, in this phase the GA tries to locate good solutions which will be fine tuned in the later phase of the algorithm. Naturally, the time which the GA will spend exploring depends on the initialisation strategy that is used. The RND strategy requires the largest amount of time to obtain good solutions, which is expected since all of them are randomly generated. On the other hand the CMB and SGP initialisation strategies can be seen to start with very good solutions, and thus the algorithm does not have to invest too much time into obtaining good solutions.

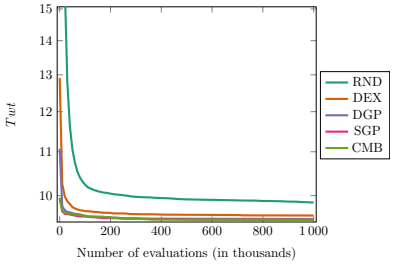
After these good solutions were obtained, the GA focuses on exploiting and further improving those solutions. This part of the evolution can best be observed on the zoomed in figures. For example, with the RND initialisation strategy the GA can be seen to still improve the solutions when it comes close the maximum number of function evaluations. Therefore, if the algorithm was given more time, it would continue to improve these solutions since it is still relatively far from the best possible solutions. However, when using other initialisation strategies it is evident that the quality of the best solution starts to stagnate before the GA reaches the maximum number of function evaluations. Only for the DEX strategy the GA does not start to stagnate for all population sizes, since it starts with a population that is inferior to that of the other three



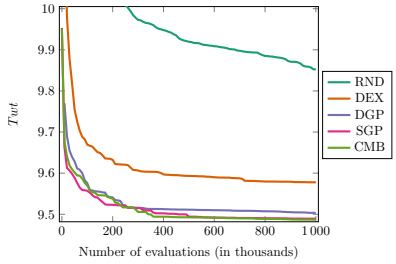
(a) Default population sizes



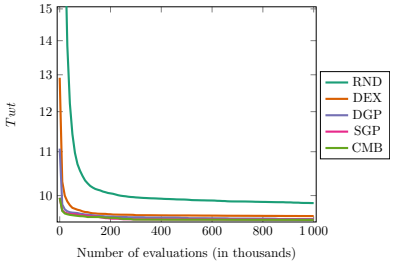
(b) Default population sizes (zoomed in)



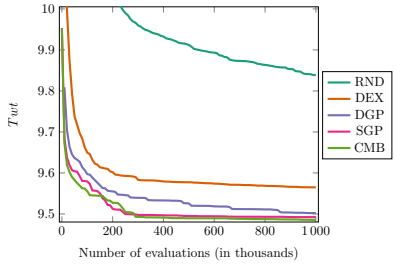
(c) Population size of 150 individuals



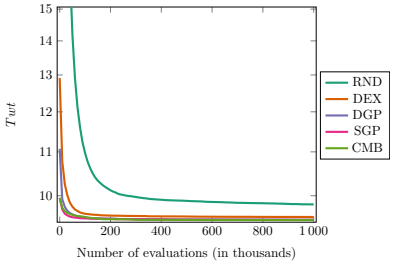
(d) Population size of 150 individuals (zoomed in)



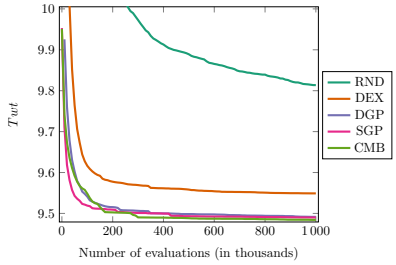
(e) Population size of 200 individuals



(f) Population size of 200 individuals (zoomed in)

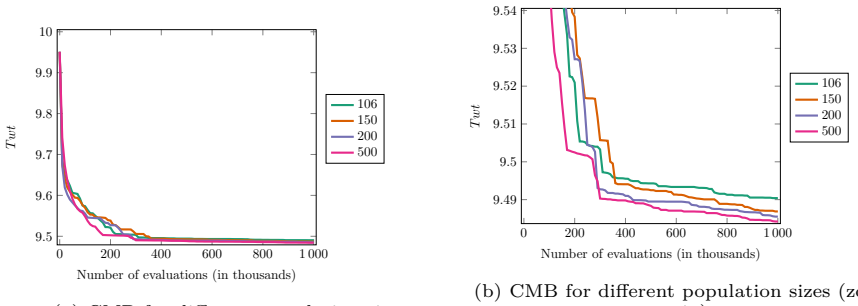


(g) Population size of 500 individuals



(h) Population size of 500 individuals (zoomed in)

Figure 5: The change of the minimum fitness value during the execution of the algorithm with different population sizes



(a) CMB for different population sizes

(b) CMB for different population sizes (zoomed in)

Figure 6: Comparison of the minimum values obtained by different population sizes for the CMB initialisation strategy

strategies. On the other hand, with the CMB and SGP initialisation strategies the GA begins to stagnate quite early in the evolution process. Therefore, with this initialisation strategy the GA could have been terminated much sooner without any significant influence on the obtained results. The other two initialisation strategies, DEX and DGP, usually start to stagnate at a later moment in the evolution, but still the GA could be terminated sooner without a significant loss in the quality of the obtained results. Therefore, these initialisation strategies do not only obtain better results, they also significantly improve the convergence speed of the GA, allowing it to reach good solutions in a much smaller amount time.

Figure 6 represents the change of the minimum fitness of the GA with the number of function evaluations for the different population sizes. The figure denotes the results for the CMB initialisation method because it obtained the best results out of all tested initialisations strategies. As the figure shows, the GA improves the solutions very little in the later phases of the evolution process, meaning that the GA converged to a good solution which it now only fine tunes. Additionally, it can be seen that by using larger population sizes the GA indeed obtains better results. However, it must be noted that these differences are almost negligible which can be seen from the scale of the graph. Therefore, increasing the population size when using the CMB initialisation strategy will not lead to a significant increase in the performance of the GA.

6. Discussion

The results in the last section have clearly demonstrated that by using DRs to initialise the initial population leads to a better performance of the GA. Such a behaviour is expected since the DRs are able to construct initial solutions of good quality. Thus, the GA does not have to invest time to generate these solutions by itself, but can rather focus on refining those solutions which were generated by the DRs. It could seem that by using solutions generated by DRs could lead to a premature convergence of the GA to a local optimum, or

that the initial population would not provide the GA with enough diversity to perform the search. However, the results show that no such thing happens, since even if larger population sizes with additional randomly generated DRs were used, there was usually no significant difference in the results. Therefore, even smaller populations which consist exclusively out of individuals generated by DRs provide enough diversity for the GA. A further proof of the superiority of the GA which uses DRs for solution initialisation of random initialisation is that in almost all cases the best solution obtained by the RND strategy was worse than the worst solution obtained by the other initialisation strategies. This just confirms that the solution distributions obtained by the RND and the other strategies significantly differ from each other.

The choice of the initialisation strategy has been demonstrated to influence the performance of the GA algorithm. Table 3 denotes the performance of the DRs, used in the different initialisation strategies, on the entire problem set. From the table it is evident that the manually generated DRs obtain the worst results among the four DR groups. Aside from obtaining the worst results, it is also evident that these rules obtain quite dispersed results. This is expected since not all DRs that were used were designed for minimising the Twt criterion. The GA also obtains inferior results when using this initialisation strategy than the other three. As a result, manually designed DRs are clearly least suitable for initialising the population of the GA. The automatically generated DRs achieve a much better performance on the problem set, which then also causes the GA to perform better. Even here it can be seen that there is a great difference between the solutions obtained by DRs designed for dynamic and static conditions. This means that DRs designed by GP for static scheduling seem to be most suitable for initialising the population of the GA. The combination of all the DRs obtains a worse median value than the DRs designed only for static scheduling, which is expected since it includes some bad results generated by manually designed DRs. This combination of DRs generates solutions with the most diversity. By using this combination of DRs the GA obtained the best results, although they were not significantly better than those obtained when using automatically designed DRs for static conditions. The additional variability that the combination of DRs provides seems to be only of little use to the GA. Therefore, for the GA it seems to be more important to generate an initial population consisting out of good solutions, than by providing more variability in the population. Even though the performances of the DRs on their own are vastly different, these differences are largely reduced after the optimisation with the GA. Thus, it would seem that all the DRs provide solutions which contain good building blocks that can be then combined and improved by the GA.

Another interesting thing that was observed from the results in the last section was the faster convergence when using the proposed initialisation strategies. Although it was expected that the GA will have a faster convergence by using the proposed initialisation strategies, it is still surprising that the differences would be so significant. The DRs do not seem to lead to a better convergence only due to the fact that they provide initial solutions of better quality. This is evident when comparing the random initialisation with the DGP and DEX

Table 3: Performance of the DRs on the problem instances

DR type	Min	Med	Max
Manual dynamic DRs (DEX)	13.30	17.10	364.0
GP generated dynamic DRs (DGP)	12.96	13.60	16.33
GP generated static DRs (SGP)	11.02	11.64	13.53
Combination of all DRs (CMB)	11.02	13.50	364.0

initialisation strategies. As was seen in the last section, GA with random initialisation can reach the quality of the initial population of the DEX and DGP strategies quite quickly. But regardless of that, GA with random initialisation was unable to reach the results that are close to those obtained when using the DEX and DGP. Therefore, it seems that the fitness of the initial population is not the only factor which influences the convergence of the GA. It is likely that the initial solutions that are provided by DRs contain elements that are also present in the optimal solution. The GA can then combine these elements and converge much faster to better solutions. On the other hand, with the random initialisation method the GA seems to converge more often to a local minimum in which it gets stuck. Since it has no representative solutions which would guide the search, the algorithm just randomly searches for good solutions which do not necessarily have to be close to the optimal solution.

Table 4 represents the number of function evaluations required by the GA with random initialisation to obtain a solution that can outperform the best solution in the initial populations generated by the other initialisation strategies. The table shows that when using either the DEX or DGP the GA with random initialisation needs up to around 100 000 function evaluations to reach the quality of the best solution in those initial populations. However, for the other two initialisation strategies the difference is more evident. For the default population sizes the GA with random initialisation needs over 600 000 function evaluations to obtain a solution better then the best one in the initial population of the SGP and CMB initialisation strategies. This means that the GA with random initialisation spends more than half of its evolution process trying to obtain a solution that is at least as good as the one in the initial population of the other two initialisation strategies. When the population size increases, the number of evaluations required to reach those solutions decreases, but still the GA with random initialisation spends around a third of its time to reach equally good solutions. This just shows how good solutions the DR initialisation strategies can provide and how much time the GA can save if it uses them instead of evolving them by itself.

To better outline the superiority of the initialisation strategies over random initialisation, Table 5 denotes the number of function evaluations required for the GA with the proposed initialisation strategies to obtain a better result than the best one obtained by the GA with random initialisation. This table

Table 4: Number of evaluations (in thousands) required for the GA with random initialisation to obtain a solution equal to the best solution in the initial population of other initialisation strategies

	Population size			
	-	150	200	500
DEX	20	30	40	60
DGP	40	50	60	100
SGP	630	380	320	320
CMB	630	390	330	330

Table 5: Number of evaluations (in thousands) required by the GA with the initialisation strategies to find a solution better than the best solution found by GA with random initialisation

	Population size			
	-	150	200	500
DEX	50	40	40	50
DGP	10	10	10	20
SGP	10	10	10	10
CMB	10	10	10	10

shows that initialisation strategies like CMB and SGP needed no more than 10 000 function evaluations to obtain a better solution than the GA with random initialisation would obtain after the entire evolution process. This means than in only 1% of its execution time, the GA with the SGP and CMB initialisation strategies obtains better solutions than the GA with random initialisation. For example, for the given parameters one execution of the GA on all problem instances took around 500 seconds with the RND strategy. If the SGP strategy were used, this would mean that the GA could obtain equally good results in only 5 seconds of execution. This observation additionally shows how much the initialisation strategies can have an influence on the convergence speed of the GA, since it can be concluded that they increased the convergence speed by more than two orders of magnitude. Such an improvement can be very important if the algorithm is used for scheduling problems in which the parameters change often and therefore it is required to quickly reconstruct the schedule.

One could argue that by using the initialisation strategies the execution time of the GA increases. However, the DRs construct the solutions in a small amount of time. For example, manually designed DRs require 0.14 seconds on average to construct a solution. On the other hand DRs generated by GP for dynamic conditions require 0.091 seconds, while DRs generated for static conditions require 0.16 seconds. Therefore, to construct the entire population, all initialisation strategies require around 5 seconds. Since it was previously

denoted that the SGP method needs around 5 seconds to reach equally good results as the RND strategy after 500 seconds, this would mean that together with the population initialisation process SGP would require around 10 seconds to reach equally good solutions as the RND strategy. Therefore, the SGP method would still be 50 times faster in obtaining equally good solutions. This large difference in the processing times just additionally denotes the large gap in the convergence speed between the RND and the other initialisation strategies.

One possible disadvantage of the initialisation methods could be that it is required to collect the existing DRs, which might be scarce for the given scheduling problem. However, since automatically generated DRs have demonstrated to obtain even better solutions, and the GA also obtains better results when using them for initialising the population, there is no need to use manually designed DRs. Naturally, automatically generated DRs need to be generated by GP or some other optimisation method, but this has already been a thoroughly researched topic, since DRs were generated for many different scheduling problems and conditions. Even though the process of generating new DRs is time consuming, it can be performed at any time prior to using them for the initialisation process. Therefore, a sufficient number of rules can be generated offline up front, and then just reused to create the initial population any time it is required. Therefore the time required to generate the DRs does not directly influence the execution time of the GA.

It is also interesting to analyse the diversity of the initial population which is constructed by each initialisation method. However, measuring the diversity between solutions is quite difficult. Therefore several measures which give a notion about the difference between the solutions are defined. All the metrics are calculated on a pair of schedules created for the same problem instance. In this paper four diversity measures will be defined: machine dissimilarity, job dissimilarity, job distance, and real job distance. The machine dissimilarity metric is calculated for two schedules as the number of jobs which are scheduled on different machines divided by the total number of jobs. For example, the value of this metric for the schedules in Figure 7 would be 0.4, since jobs 0, 2, 6, and 7 are allocated to different machines in those two schedules. On the other hand, job dissimilarity measures the diversity of the sequences of jobs on the machines. Since it is difficult to measure the difference in job sequences if certain jobs are allocated to different machines, this measure only considers jobs which are allocated to the same machine. This measure is calculated as the number of jobs that are on the same position on a machine in both schedules, divided by the number of jobs that are allocated to the same machines in both schedules. In the example, jobs 1, 3, 5, and 9 are allocated to machine 1 in both of schedules, while jobs 4 and 8 are allocated to machine 2. For machine 1 their ordering in schedule 1 is 3, 9, 5, 1, while their ordering in schedule 2 is 3, 1, 9, 5. This means that job 3 has the same position in both schedules, while the other jobs have different positions, and thus this measure for this machine would be 0.75, since three out of the four jobs are located on different positions in two schedules. For the entire schedule the value of this measure would be 0.83, since the two jobs allocated to machine 2 in both schedules are on different

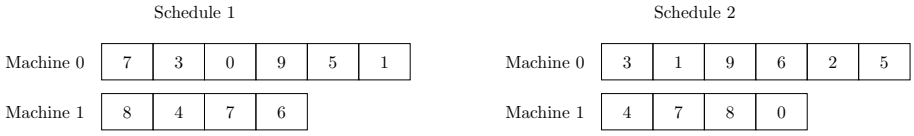


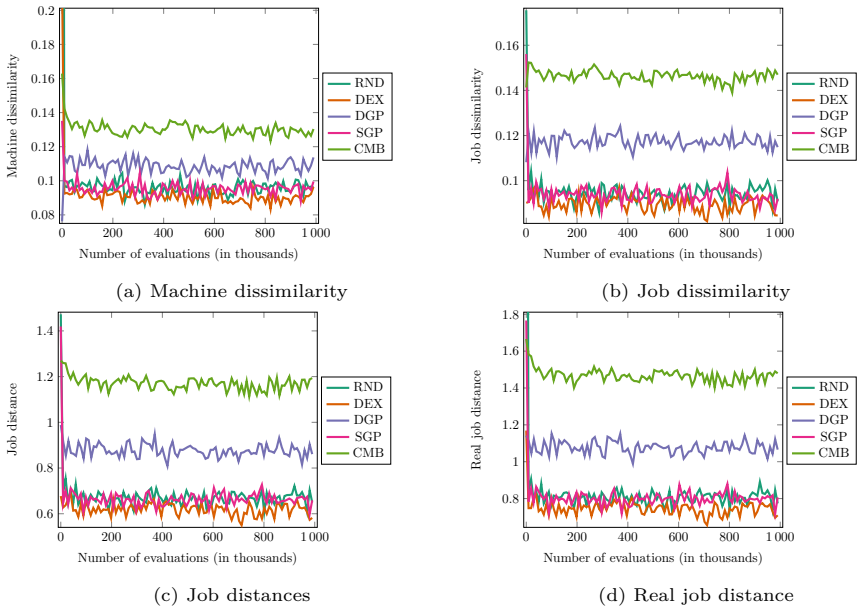
Figure 7: Example of two schedules

positions. The job distance measure is calculated in a similar way, but instead of measuring the number of jobs on different positions, it sums the distance of their positions on the machine in different schedules. For example, since job 3 is on the same position in both schedules, its distance would be 0, while for job 1 the distance would be 2, because on machine 1 it is located on position 4, but on machine 2 it is located on position 2. As is evident from the description, all jobs which are not located on the same machine in both schedule are not considered when calculating the distance. This can be a bit misleading, since it is possible that between two jobs there are a lot of jobs that are in between but are not allocated to the same machine in both schedules. Therefore, another measure denoted as real job distance is defined. The value of this metric for job 1 would be 4 since the position of the job would be 6 on machine 1 if all jobs were considered. The metric values in the following paragraphs represent the average metric values calculated between each pair of schedules constructed by each of the initialisation methods.

Table 6 denotes the values of the diversity measures for the starting population when the default population sizes are used. The table shows that there is a big difference in the diversity of the starting population when different initialisation techniques are used. The most diverse solutions are obtained by the random initialisation method, which is evident from the fact that it obtained the largest vales for all the measures. The difference between it and the other methods is especially large for the machine dissimilarity measure, which means that a large number of jobs are allocated to different machines in the generated schedules, and the real job distance. This is expected since the jobs are allocated completely randomly to the machines. On the other hand, the initialisation methods which used existing DRs all generate less diverse initial populations. In the schedules constructed by these methods jobs are more likely to be scheduled to the same machine, and will usually be scheduled in similar sequences. The DEX initialisation method has the largest value for the machine dissimilarity measure, which is expected since the DRs in this initialisation method optimise different criteria, therefore it is expected that they will schedule jobs to different machines. On the other hand, DEX and DGP have the smallest job dissimilarity, which is the consequence of the fact that they are dynamic rules and thus schedule jobs mostly in the order of their arrival. Figure 8 shows the change of the diversity metrics during the evolution process. From the figure it is evident that the diversity metrics quickly decrease at the start of the evolution process, until they reach a certain value which remains similar during the entire evolution process. The RND and DGP initialisation methods have the largest

Table 6: Diversity metric values for the initial populations constructed by the population initialisation methods with the default population sizes

	RND	DEX	DGP	SGP	CMB
Machine dissimilarity	80.05	21.52	7.62	13.52	16.29
Job dissimilarity	17.54	9.02	10.82	15.61	14.13
Job distance	1.473	0.677	0.988	1.419	1.270
Real job distance	5.495	1.169	1.142	1.766	1.666



(a) Machine dissimilarity

(b) Job dissimilarity

(c) Job distances

(d) Real job distance

Figure 8: Diversity metric dynamics during evolution with default population sizes

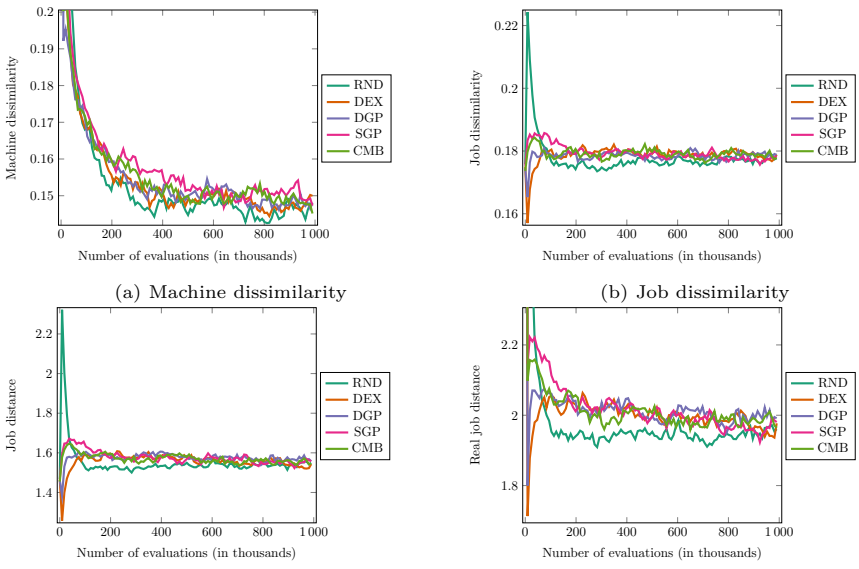
values for the diversity measures during the entire evolution process, while the other methods usually have similar values for all the metrics.

Table 7 shows the values of the metrics when the population size of 500 individuals is used. This time the difference in the diversity metrics between the different initial populations is almost non existent. The reason for this is that now most of the population is initialised randomly. Therefore, in this case the initialisation method does not have any influence on the diversity of the solutions. Figure 9 shows the dynamics of the diversity metrics during the evolution process. In this case the dynamics and values of the diversity metrics are similar for all five population initialisation methods.

Based on the population diversity analysis, it can be concluded that the initialisation methods have a large influence on the population diversity if the population is only initialised by them. However, if only a part of the population

Table 7: Diversity metric values for the initial populations constructed by the population initialisation methods with the population size of 500 individuals

	RND	DEX	DGP	SGP	CMB
Machine dissimilarity	80.00	79.84	79.30	79.77	77.17
Job dissimilarity	17.58	17.51	17.45	17.53	17.36
Job distance	1.468	1.464	1.461	1.466	1.455
Real job distance	5.468	5.435	5.400	5.430	5.258



(a) Machine dissimilarity

(b) Job dissimilarity

(c) Job distances

(d) Real job distance

Figure 9: Diversity metric dynamics during evolution with a population of 500 individuals

is initialised by DRs and most of it is initialised randomly, then the diversity of the initial population is quite similar. Since the previous sections showed that there is mostly no significant difference when increasing the population size, it can be concluded that the diversity of the initial population does not have a critical influence on the performance of the GA. Rather, it seems that the most important factor is the quality of the initial solutions that are used.

All the provided results demonstrated that population initialisation with DRs has significantly improved the performance of the GA, while requiring a significantly smaller amount of time to reach good solutions. The method as it is could be used for solving any real world unrelated machines scheduling problem without any significant modifications. However, the initialisation procedure with DRs makes GAs even more competitive for real world scheduling problems. Since the GA can now obtain much better results, this makes it more desirable for being used in real world applications. Furthermore, due to its fast convergence, the GA is also more likely to be used for problems which need to be solved under a certain time constraint. As a result, the GA could be applied in problems which are performance and time critical. Therefore, it is safe to conclude that the simple initialisation procedure leads to a significant improvement on the performance of the algorithm.

7. Conclusion

This paper analysed the influence of initialising the starting population of a GA with solutions generated by various DRs. In comparison with random initialisation, by using the DR based initialisation strategies the GA obtained significantly better results. Apart from providing better results, the initialisation strategies also improve the convergence rate of the algorithm, allowing it to reach better solutions in a smaller amount of function evaluations. Furthermore, by using the initialisation strategies based on DRs the GA obtains more stable results, which reduces the need to execute the GA several times to ensure that good solutions are obtained, and that the GA did not get stuck in a local optimum. Finally, as the execution time of DRs is almost negligible, using them for the construction of initial solutions does not have a large effect on the total execution time of the GA.

Out of the four tested initialisation strategies based on DRs, the best results were obtained by the CMB strategy which used all the DRs together, and the SGP strategy which used the only DRs suitable for scheduling under static conditions which were generated by GP. Since there is no major difference between the results obtained by CMB and SGP, it is sufficient to use only automatically generated DRs for initialising the population of the GA. This has a large benefit since GP can be used to design DRs for various scheduling problems and can be used to generate any number of DRs.

Depending on the initialisation method the improvements ranged up to 4% on average over the random initialisation method. The time to obtain the same quality of the results was at the same time largely reduced, up to a factor of 50. Furthermore, the GA has proven to produce more stable results, meaning

that upon several executions the spread of the obtained results is minimal. The reduced population diversity of the populations initialised by the DRs did not influence the performance of the GA, meaning that the diversity is not necessarily the main driving force in this case, but rather the availability of good solutions in the population. Another great benefit of DRs is that unlike other initialisation strategies, they do not have to be designed manually, but can rather be generated with the use of GP, which reduces the effort that has to be invested into the design of the algorithm. Since all the obtained results speak in favour of the DR initialisation methods that these strategies significantly improve the GA for solving the unrelated machines scheduling problem. Therefore, such strategies should be used as a part of GAs in order to improve their performance and convergence.

Since the GA with the population initialised by using DRs achieves great improvements over the GA which uses randomly generated DRs, there is motivation to research this topic further in future studies. One direction would be to investigate scheduling problems which have additional constraints like setup times, precedence constraints, etc. Additionally, another direction would be to test all the described initialisation strategies in other scheduling environments, to see how they would influence the performance of the GA for solving those scheduling problems. Finally, it would also be interesting to investigate the effect of such initialisation strategies when performing multi-objective optimisation.

Declaration of interest and funding

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Declarations of interest: none

References

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`.

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.

Boussaïd, I., Lepagnot, J., & Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences*, *237*, 82 – 117. URL: http://www.sciencedirect.com/science/article/pii/S0020025513001588. doi:https://doi.org/10.1016/j.ins.2013.02.041. Prediction, Control and Diagnosis using Advanced Neural Computations.

Branke, J., Hildebrandt, T., & Scholz-Reiter, B. (2015). Hyper-heuristic Evolution of Dispatching Rules: A Comparison of Rule Representations. *Evolutionary Computation*, *23*, 249–277. URL: http://www.mitpressjournals.org/doi/10.1162/EVCO_a_00131. doi:10.1162/EVCO_a_00131.

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.

Branke, J., & Pickardt, C. W. (2011). Evolutionary search for difficult problem instances to support the design of job shop dispatching rules. *European Journal of Operational Research*, *212*, 22–32. URL: http://linkinghub.elsevier.com/retrieve/pii/S0377221711000981. doi:10.1016/j.ejor.2011.01.044.

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.

Burke, E. K., Newall, J. P., & Weare, R. F. (1998). Initialization strategies and diversity in evolutionary timetabling. *Evol. Comput.*, *6*, 81–103. URL: http://dx.doi.org/10.1162/evco.1998.6.1.81. doi:10.1162/evco.1998.6.1.81.

Cheng, R., Gen, M., & Tsujimura, Y. (1996). A tutorial survey of job-shop scheduling problems using genetic algorithms—i. representation. *Computers & Industrial Engineering*, *30*, 983–997. URL: https://doi.org/10.1016/0360-8352(96)00047-2. doi:10.1016/0360-8352(96)00047-2.

Cheng, R., Gen, M., & Tsujimura, Y. (1999a). A tutorial survey of job-shop scheduling problems using genetic algorithms, part II: hybrid genetic search strategies. *Computers & Industrial Engineering*, *36*, 343–364. URL: https://doi.org/10.1016/s0360-8352(99)00136-9. doi:10.1016/s0360-8352(99)00136-9.

Cheng, V., Crawford, L., & Menon, P. (1999b). Air traffic control using genetic search techniques. In *Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No.99CH36328)* (pp. 249–254). IEEE volume 1. URL: http://ieeexplore.ieee.org/document/806209/. doi:10.1109/CCA.1999.806209.

Chiang, T. C., Shen, Y. S., & Fu, L. C. (2008). A new paradigm for rule-based scheduling in the wafer probe centre. *International Journal of Production Research*, *46*, 4111–4133. URL: https://doi.org/10.1080/00207540601137199. doi:10.1080/00207540601137199.

Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Distributed optimization by ant colonies.

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.

Diaz-Gomez, P. A., & Hougen, D. F. (2007). Initial population for genetic algorithms: A metric approach. In *GEM*.

Dimopoulos, C., & Zalzala, A. (2000). Recent developments in evolutionary computation for manufacturing optimization: problems, solutions, and comparisons. *IEEE Transactions on Evolutionary Computation*, *4*, 93–113. URL: http://ieeexplore.ieee.org/document/850651/. doi:10.1109/4235.850651.

Durasević, M., & Jakobović, D. (2018). Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment. *Genetic Programming and Evolvable Machines*, *19*, 9–51. URL: https://doi.org/10.1007/s10710-017-9310-3. doi:10.1007/s10710-017-9310-3.

Ernst, A., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, *153*, 3–27. URL: https://doi.org/10.1016/s0377-2217(03)00095-x. doi:10.1016/s0377-2217(03)00095-x.

Fitzgerald, J. M., Ryan, C., Medernach, D., & Krawiec, K. (2015). An integrated approach to stage 1 breast cancer detection. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation* GECCO '15 (pp. 1199–1206). New York, NY, USA: ACM. URL: http://doi.acm.org/10.1145/2739480.2754761. doi:10.1145/2739480.2754761.

Gandomi, A. H., & Yang, X.-S. (2014). Chaotic bat algorithm. *Journal of Computational Science*, *5*, 224 – 232. doi:https://doi.org/10.1016/j.jocs.2013.10.002. Empowering Science through Computing + BioInspired Computing.

Gao, J., Gen, M., Sun, L., & Zhao, X. (2007). A hybrid of genetic algorithm and bottleneck shifting for multiobjective flexible job shop scheduling problems. *Computers & Industrial Engineering*, *53*, 149–162. URL: https://doi.org/10.1016/j.cie.2007.04.010. doi:10.1016/j.cie.2007.04.010.

Geem, Z. W., Kim, J. H., & Loganathan, G. (2001). A new heuristic optimization algorithm: Harmony search. *SIMULATION*, *76*, 60–68. doi:10.1177/003754970107600201.

Gogna, A., & Tayal, A. (2013). Metaheuristics: review and application. *Journal of Experimental & Theoretical Artificial Intelligence*, *25*, 503–526. URL: https://doi.org/10.1080/0952813X.2013.782347. doi:10.1080/0952813X.2013.782347. arXiv:https://doi.org/10.1080/0952813X.2013.782347.

Han, Y., Gong, D., Li, J., & Zhang, Y. (2016). Solving the blocking flow shop scheduling problem with makespan using a modified fruit fly optimisation algorithm. *International Journal of Production Research*, *54*, 6782–6797. doi:10.1080/00207543.2016.1177671.

Hansen, J. V. (2004). Genetic search methods in air traffic control. *Computers & Operations Research*, *31*, 445–459. URL: http://linkinghub.elsevier.com/retrieve/pii/S0305054802002289. doi:10.1016/S0305-0548(02)00228-9.

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.

Hou, E., Ansari, N., & Ren, H. (1994). A genetic algorithm for multiprocessor scheduling. *IEEE Transactions on Parallel and Distributed Systems*, *5*, 113–120. URL: https://doi.org/10.1109/71.265940. doi:10.1109/71.265940.

Ishibuchi, H., Yoshida, T., & Murata, T. (2003). Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation*, *7*, 204–223. URL: https://doi.org/10.1109/tevc.2003.810752. doi:10.1109/tevc.2003.810752.

Kaboli, H. R., Fallahpour, A., Kazemi, N., Selvaraj, J., & Abd Rahim, N. (2016a). An expression-driven approach for long-term electric power consumption forecasting. *American Journal of Data Mining and Knowledge Discovery*, *x, No. x*, 1–13.

Kaboli, S. H. A., & Alqallaf, A. K. (2019). Solving non-convex economic load dispatch problem via artificial cooperative search algorithm. *Expert Systems with Applications*, *128*, 14 – 27. doi:https://doi.org/10.1016/j.eswa.2019.02.002.

Kaboli, S. H. A., Fallahpour, A., Selvaraj, J., & Rahim, N. (2017a). Long-term electrical energy consumption formulating and forecasting via optimized gene expression programming. *Energy*, *126*, 144 – 164. URL: http://www.sciencedirect.com/science/article/pii/S0360544217303675. doi:https://doi.org/10.1016/j.energy.2017.03.009.

Kaboli, S. H. A., Selvaraj, J., & Rahim, N. (2016b). Long-term electric energy consumption forecasting via artificial cooperative search algorithm. *Energy*, *115*, 857 – 871. doi:https://doi.org/10.1016/j.energy.2016.09.015.

Kaboli, S. H. A., Selvaraj, J., & Rahim, N. (2017b). Rain-fall optimization algorithm: A population based algorithm for solving constrained optimization problems. *Journal of Computational Science*, *19*, 31 – 42. doi:https://doi.org/10.1016/j.jocs.2016.12.010.

Kazimipour, B., Li, X., & Qin, A. K. (2013). Initialization methods for large scale global optimization. In *2013 IEEE Congress on Evolutionary Computation* (pp. 2750–2757). doi:10.1109/CEC.2013.6557902.

Kazimipour, B., Li, X., & Qin, A. K. (2014). A review of population initialization techniques for evolutionary algorithms. In *2014 IEEE Congress on Evolutionary Computation (CEC)* (pp. 2585–2592). doi:10.1109/CEC.2014.6900618.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks* (pp. 1942–1948 vol.4). volume 4. doi:10.1109/ICNN.1995.488968.

Kofler, M., Wagner, S., Beham, A., Kronberger, G., & Affenzeller, M. (2009). Priority Rule Generation with a Genetic Algorithm to Minimize Sequence Dependent Setup Costs. In R. Moreno-Díaz, F. Pichler, & A. Quesada-Arencibia (Eds.), *Computer Aided Systems Theory - EUROCAST 2009: 12th International Conference, Las Palmas de Gran Canaria, Spain, February 15-20, 2009, Revised Selected Papers* (pp. 817–824). Berlin, Heidelberg: Springer Berlin Heidelberg. URL: http://link.springer.com/10.1007/978-3-642-04772-5_105. doi:10.1007/978-3-642-04772-5_105.

Krawiec, K., & Pawlak, M. (2015). Genetic programming with alternative search drivers for detection of retinal blood vessels. doi:10.1007/978-3-319-16549-3_45.

Kuczapski, A., Micea, M., Maniu, L., & Cretu, V. (2010). Efficient generation of near optimal initial populations to enhance genetic algorithms for job-shop scheduling, . *39*, 32–37.

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.

Lee, Y. H., Bhaskaran, K., & Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. *IIE Transactions*, *29*, 45–52. URL: http://www.tandfonline.com/doi/abs/10.1080/07408179708966311. doi:10.1080/07408179708966311.

Li, J., Pardalos, P. M., Sun, H., Pei, J., & Zhang, Y. (2015). Iterated local search embedded adaptive neighborhood selection approach for the multi-depot vehicle routing problem with simultaneous deliveries and pickups. *Expert Systems with Applications*, *42*, 3551 – 3561. doi:https://doi.org/10.1016/j.eswa.2014.12.004.

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.

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.

Modiri-Delshad, M., Kaboli, S. H. A., Taslimi-Renani, E., & Rahim, N. A. (2016). Backtracking search algorithm for solving economic dispatch problems with valve-point effects and multiple fuel options. *Energy*, *116*, 637 – 649. doi:https://doi.org/10.1016/j.energy.2016.09.140.

Mohamad Izdin Hlal, A., K. Ramachandaramurthya, V., Sanjeevikumar, P., Pouryekta, A., Kaboli, H. R., & bin Tuan Abdullah, T. A. R. (2019). Nsga-ii and mopso based optimization for sizing of hybrid pv / wind / battery energy storage system. *International Journal of Power Electronics and Drive Systems*, *10*, 463–478. doi:10.11591/ijpeds.v10n1.pp463-478.

Nguyen, S., Mei, Y., & Zhang, M. (2017). Genetic programming for production scheduling: a survey with a unified framework. *Complex & Intelligent Systems*, *3*, 41–66. URL: http://link.springer.com/10.1007/s40747-017-0036-x. doi:10.1007/s40747-017-0036-x.

Petrovic, S., & Castro, E. (2011). A genetic algorithm for radiotherapy pretreatment scheduling. In C. Di Chio, A. Brabazon, G. A. Di Caro, R. Drechsler, M. Farooq, J. Grahl, G. Greenfield, C. Prins, J. Romero, G. Squillero, E. Tarantino, A. G. B. Tettamanzi, N. Urquhart, & A. Ş. Uyar (Eds.), *Applications of Evolutionary Computation: EvoApplications 2011: EvoCOMNET,*

EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, April 27-29, 2011, Proceedings, Part II (pp. 454–463). Berlin, Heidelberg: Springer Berlin Heidelberg.

Pfund, M. E., Mason, S. J., & Fowler, J. W. (2006). Semiconductor Manufacturing Scheduling and Dispatching. In *Handbook of Production Scheduling* (pp. 213–241). Boston: Kluwer Academic Publishers. URL: http://link.springer.com/10.1007/0-387-33117-4_9. doi:10.1007/0-387-33117-4_9.

Picek, S., Cupic, M., & Rotim, L. (2016). A new cost function for evolution of s-boxes. *Evolutionary Computation*, *24*, 695–718. doi:10.1162/EVCO_a_00191.

Picek, S., Jakobovic, D., Miller, J. F., Batina, L., & Cupic, M. (2016). Cryptographic boolean functions: One output, many design criteria. *Applied Soft Computing*, *40*, 635 – 653. doi:https://doi.org/10.1016/j.asoc.2015.10.066.

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.

Rahnamayan, S., Tizhoosh, H. R., & Salama, M. M. (2007a). A novel population initialization method for accelerating evolutionary algorithms. *Computers and Mathematics with Applications*, *53*, 1605 – 1614. URL: http://www.sciencedirect.com/science/article/pii/S0898122107001344. doi:https://doi.org/10.1016/j.camwa.2006.07.013.

Rahnamayan, S., Tizhoosh, H. R., & Salama, M. M. A. (2007b). A novel population initialization method for accelerating evolutionary algorithms. *Comput. Math. Appl.*, *53*, 1605–1614. URL: http://dx.doi.org/10.1016/j.camwa.2006.07.013. doi:10.1016/j.camwa.2006.07.013.

Sarathambekai, S., & Umamaheswari, K. (2017). Intelligent discrete particle swarm optimization for multiprocessor task scheduling problem. *Journal of Algorithms & Computational Technology*, *11*, 58–67. doi:10.1177/1748301816665521.

Sebtahmadi, S. S., Azad, H. B., Kaboli, S. H. A., Islam, M. D., & Mekhilef, S. (2018). A pso-dq current control scheme for performance enhancement of z-source matrix converter to drive im fed by abnormal voltage. *IEEE Transactions on Power Electronics*, *33*, 1666–1681.

Singh, S., & Chana, I. (2016). A survey on resource scheduling in cloud computing: Issues and challenges. *Journal of Grid Computing*, *14*, 217–264. URL: https://doi.org/10.1007/s10723-015-9359-2. doi:10.1007/s10723-015-9359-2.

Soler-Dominguez, A., Juan, A. A., & Kizys, R. (2017). A survey on financial applications of metaheuristics. *ACM Comput. Surv.*, *50*, 15:1–15:23. URL: http://doi.acm.org/10.1145/3054133. doi:10.1145/3054133.

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*, *113*, 555 – 569. URL: http://www.sciencedirect.com/science/article/pii/S0957417418304159. doi:https://doi.org/10.1016/j.eswa.2018.06.053.

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.

Vigneswari, T., & Mohamed, D. M. A. M. (2014). Performance analysis of initialization methods for optimizing artificial bee colony grid scheduling.

Wang, L., Siegel, H. J., Roychowdhury, V. P., & Maciejewski, A. A. (1997). Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach. *Journal of Parallel and Distributed Computing*, *47*, 8–22. URL: https://doi.org/10.1006/jpdc.1997.1392. doi:10.1006/jpdc.1997.1392.

Xhafa, F., & Abraham, A. (2008). Meta-heuristics for grid scheduling problems. In F. Xhafa, & A. Abraham (Eds.), *Metaheuristics for Scheduling in Distributed Computing Environments* (pp. 1–37). Berlin, Heidelberg: Springer Berlin Heidelberg.

Xiong, J., Tan, X., Yang, K.-w., Xing, L.-n., & Chen, Y.-w. (2012). A hybrid multiobjective evolutionary approach for flexible job-shop scheduling problems. *Mathematical Problems in Engineering*, *2012*. URL: http://dx.doi.org/10.1155/2012/478981. doi:10.1155/2012/478981.

Yang, S., Guohui, Z., Liang, G., & Kun, Y. (2009). A novel initialization method for solving flexible job-shop scheduling problem. In *2009 International*

Conference on Computers Industrial Engineering (pp. 68–73). doi:10.1109/ICCIE.2009.5223891.

Zhan, Z.-H., Liu, X.-F., Gong, Y.-J., Zhang, J., Chung, H. S.-H., & Li, Y. (2015). Cloud computing resource scheduling and a survey of its evolutionary approaches. *ACM Computing Surveys*, *47*, 1–33. URL: https://doi.org/10.1145/2788397. doi:10.1145/2788397.

Zhou, H., Feng, Y., & Han, L. (2001). The hybrid heuristic genetic algorithm for job shop scheduling. *Computers & Industrial Engineering*, *40*, 191–200. URL: https://doi.org/10.1016/s0360-8352(01)00017-1. doi:10.1016/s0360-8352(01)00017-1.