Adaptive scheduling on unrelated machines with genetic programming

Marko Đurasević, Domagoj Jakobović, Karlo Knežević

University of Zagreb, Faculty of Electrical Engineering and Computing, Croatia

marko.durasevic@fer.hr, domagoj.jakobovic@fer.hr, karlo.knezevic@fer.hr

Abstract

This paper investigates the use of genetic programming in automatized synthesis of heuristics for the parallel unrelated machines environment with an arbitrary performance criteria. The proposed scheduling heuristic consists of a manually defined meta-algorithm which uses a priority function evolved separately with genetic programming. In this paper, several different genetic programming methods for evolving priority functions, like dimensionally aware genetic programming, genetic programming with iterative dispatching rules and gene expression programming, have been tried out and described. The performance of the suggested approach is compared to existing scheduling heuristics and it is shown that it mostly outperforms them. The described approach could prove useful when used for optimizing scheduling criteria for which no adequate scheduling heuristic exists.

Keywords: Scheduling on unrelated machines, Genetic programming, Priority scheduling

1. Introduction

Scheduling can be defined as a decision-making process concerned with the allocation of scarce resources to tasks over a given time period in order to optimize one or more objectives [1]. Unfortunately, most of the important scheduling objectives represent NP-hard problems. As a consequence, no efficient algorithms are available for obtaining the optimal solution for a given objective. Therefore, solutions are usually obtained by using heuristic algorithms. In this context, we divide the algorithms in two groups: the first group consists of metaheuristics which search the space of solutions (schedules) to find the best one; the second group consists of problem-specific heuristics that construct the solution using some features of the problem [2].

Since most scheduling problems are combinatorial by nature, *search-based* metaheuristic methods (such as genetic algorithms, ant colony optimization, particle swarm optimization, etc.) can be used to search the space of solutions. The solutions obtained by such methods are often of a good quality, mostly better than the solutions obtained by constructive heuristics. Unfortunately, these methods require a substantial computational time in order to obtain solutions of acceptable quality (since they search the entire solution space). Another disadvantage of these methods is that they are generally not applicable in dynamic conditions, in which a constant adaptation to the changing conditions may be needed (e.g. unplanned arrival of new jobs, machine outages etc).

Constructive scheduling heuristics, on the other hand, do not search the space of all possible solutions, but instead directly build the solution (schedule). Because of that, these heuristics can quickly react to changes in the environment, making them applicable in dynamic conditions. The advantage of these heuristics over the *search-based* methods is that their computational complexity is almost negligible. However, constructive scheduling heuristics also cope with a certain number of problems. For instance, it is often hard to select the optimal heuristic for the given criteria and problem instance. This was shown in [3] where evolutionary algorithms were used in order to create problem instances for the job shop scheduling problem, on which

E-mail addresses:

certain constructive scheduling heuristics made poor choices, which in the end resulted in schedules of low quality.

It should also be noted that such heuristics are not designed to optimize arbitrary criteria which could be defined by the user (it would be necessary to design a new heuristic to handle such a case). When all this is considered, it can be concluded that selecting an appropriate scheduling policy is not easy and that a heuristic for optimizing a given criteria might not even exist.

Genetic programming (GP), although rarely used for solving scheduling problems, is very suitable for searching the space of algorithms which, in turn, can provide solutions to scheduling problems. Recently, GP has been used to evolve scheduling policies for a wide variety of environments (single machine scheduling [4, 5, 6], job shop scheduling [7, 8, 9, 10, 11, 12, 13], parallel proportional machine scheduling [14], airplane scheduling in air traffic control [15, 16], scheduling in semiconductor manufacturing [17]). In addition, a recent survey about evolving dispatching rules with GP has been conducted by Branke *et al.* in [18]. This method has proven to be very successful, because not only does it allow to create scheduling policies for arbitrary criteria, but it also provides solutions which are on par with solutions obtained by heuristic methods.

In this paper we describe the approach of using GP for creating scheduling policies for the unrelated machines environment. We also describe several variants for this approach, used in order to obtain better results. Some of these modifications have been used in other machine environments, but to our knowledge, they have not yet been applied in the unrelated machines scheduling. We compare all of these variants against each other and additionally compare them to several existing approaches for creating schedules in order to assess the quality of the achieved solutions. In that regard this paper can be viewed as a continuation in comparing different optimization approaches, which has previously been conducted by Nguyen *et al.* [19] and Branke *et al.* [20].

The remainder of the paper is organized as follows: in Section 2 the definition of scheduling in the unrelated machines environment is given. Section 3 gives a short overview of genetic programming and its usage in the creation of scheduling policies. In Section 4 various GP optimizations are described. Section 5 describes the benchmark set and the results for all the algorithms. Section 6 delivers a short discussion on the results. Finally, Section 7 gives a short conclusion.

2. The Unrelated Machines Environment

In the unrelated machines environment, a number of n jobs compete in order to be processed on one of the m machines. All jobs have a processing time p_{ij}, which determines the time which is needed for the job with the index j to be processed on the machine with the index i, as well as a release time r_j (ready time) which determines when the job becomes available for scheduling. Jobs may also have additional properties, like a due date d_j and a weight w_j (which determines the importance of the job). In this paper we considered the more complex problem variant which includes relative importance of a job, given by its weight. Solving for this variant can also solve the simpler problem with unweighted jobs.

2.1. Scheduling Criteria

The most common scheduling criteria which are used for this environment include tardiness, number of tardy jobs, flowtime and makespan. First, let us define those criteria for a single job. Let C_j denote the finishing time of the job j. We can then define tardiness (the amount of time that a job was late) of job j as:

$$T_j = \max\{C_j - d_j, 0\}. \tag{1}$$

Similarly, flowtime, the amount of time a job spends in the system, can be defined as

$$F_j = C_j - r_j. \tag{2}$$

We will also define an additional measure which determines if a job is tardy or not

$$U_j = \begin{cases} 1 : T_j > 0 \\ 0 : T_j = 0 \end{cases}. \tag{3}$$

Using criteria for individual jobs, criteria for the entire schedule are defined. The makespan is defined as the maximum finishing time of all the jobs in the set

$$C_{max} = \max\{C_j\}. \tag{4}$$

The other criteria are often defined as weighted sums: weighted tardiness

$$T_w = \sum_j wT_j, \tag{5}$$

weighted flowtime

$$F_w = \sum_j wF_j, \tag{6}$$

and weighted number of tardy jobs

$$U_w = \sum_j wU_j. \tag{7}$$

2.2. Scheduling Conditions

Based on the availability of the job parameters, scheduling can be performed in different conditions. If all the parameters are known before the jobs are ready, then the schedule can be produced before the system starts its execution. This type of scheduling is called off-line or static scheduling. Search-based methods are most often used to create schedules for this type of scheduling conditions.

On the other hand, if no information about the jobs is available until the job has arrived into the system and no information is available about future jobs either, then such a scheduling process is called on-line scheduling. Heuristic scheduling methods, and the GP-based scheduler described in the next section, are almost always used in this kind of scheduling conditions. In addition, both of these approaches can be used for off-line scheduling as well, but in that case they are generally less efficient than the search-based methods.

3. Scheduling in the Unrelated Machines Environment Using Genetic Programming

3.1. Genetic Programming

Genetic programming (GP) [21] is an evolutionary algorithm which is used to discover functions or programs which provide a solution to a given problem. In the algorithm, these solutions are represented in the form of a tree. The tree consists of two types of nodes, namely functional nodes (which represent certain arithmetic, boolean or other kind of functions) and terminal nodes which represent input variables and constants.

The idea behind this approach is to simulate the process of evolution. At the beginning of the algorithm, a random number of potential solutions is generated. Each solution receives an estimation of how "good" the solution is, which is measured on some predefined test cases. This estimation is called the *fitness* of the solution. The algorithm simulates natural selection such that the "better" individuals (solutions) have a higher probability of survival, while on the other hand, "worse" individuals have a smaller probability to survive. The individuals which survived the selection participate in crossover, which is a genetic operator that combines two individuals to form a new, hopefully better, individual. After the new individual is created, another operator called mutation is applied to the newly created individual. This operator changes, with a certain probability, some elements of the individual in order to introduce new elements into the

solutions. This cycle of selection, crossover and mutation is repeated until a certain stopping criteria is achieved (e.g. reaching a predefined number of iterations, finding an acceptable solution).

Genetic programming can be used to solve many classes of problems, including classification [22, 23] and regression [24]. It was also used to generate human-competitive solutions in a wide area of fields [25]. The presented GP scheduling system was implemented in the ECF framework [26].

3.2. The Scheduling Procedure

Solutions to scheduling problems are usually represented as a sequence of jobs which are to be executed on each of the machines. This sequence represents the solution to only a single scheduling problem instance. For a different scheduling problem instance a new solution must be found, and most of the search-based methods work this way. On the other hand, by using genetic programming it is possible to create algorithms which will in turn be able to generate solutions for arbitrary problem instances in a given scheduling environment. This is referred to as a hyper-heuristic approach, since GP is searching the space of possible algorithms, rather than the space of possible schedules.

The scheduling method applied in this work consists of two parts: the meta-algorithm and a priority function. The priority function determines the priority of job j if scheduled on machine i; this part is evolved with GP to allow for optimization of an arbitrary scheduling criterion. The meta-algorithm *uses* a priority function to decide which job to process on which machine when the next decision needs to be made. We use a manually defined meta-algorithm, which assumes that an appropriate priority function has previously been selected (among the existing ones) or evolved for a given criterion.

3.3. Evolving Priority Functions with GP

The scheduling paradigm where the choice of the next job being run on a certain machine depends on a particular priority value is called priority scheduling. An example of a well-known priority function may be given with

$$\pi_j = w_j/p_{ij},$$

which is defined for each job j. In the above example, the higher priority will be given to jobs with larger weight and shorter processing time (weighted shortest processing time rule, WSPT). The scheduler (meta-algorithm) will use this information when deciding which job to assign to an available machine. In our approach, the goal of genetic programming is to find an *appropriate* priority function which yields the best result given a certain scheduling objective.

To be able to evolve an appropriate priority function, the GP must be able to use relevant job and machine parameters and variables that describe the current state of the system. That is why the most important preparatory step is the selection of terminal nodes (variables) that may appear in a tree representation of the priority function. Table 1 represents the set of terminal nodes used by the GP. The *time* variable, which is used in the definition of some terminals, represents the current time of the system.

As it can be seen from the table, not all terminal nodes are used when constructing priority functions for certain criteria, because not all nodes provide useful information for all criteria. The selection of these terminals is based on the available job and machine features, as well as features that appear in existing scheduling heuristics. Examples of actual evolved priority functions are given in Sections 4 and 6.

3.4. The Meta-Algorithm and Scheduling Complexity

Assigning jobs to machines is in most cases a trivial matter, but in some environments there is a need for defining a procedure which determines how jobs are scheduled based on their priority values. This is especially true in dynamic environments, where jobs arrive over time and maybe a job cannot be executed before another job finishes. Therefore, a *meta-algorithm*, which determines which jobs should be assigned to which machines (based on the priority values), must be defined for every scheduling environment. It should be noted that the priority function and the meta-algorithm are loosely coupled, meaning that the same meta-algorithm can be used with different priority functions and that the same priority function can be used with

Table 1: Terminal nodes

Node name	**Description**
pt	processing time of job j on the machine i (p_{ij})
pmin	the minimal job processing time on all machines: $\min\{p_{ij}\}\forall i$
pavg	the average processing time on all machines
PAT	patience - the amount of time until the machine with the minimal processing time for the current job will be available
MR	machine ready - the amount of time until the current machine becomes available
age	the time that the job spent in the system: $time - r_j$
Terminals used only for due date related criteria (tardiness and number of tardy jobs)	
dd	due date (d_j)
w	weight (w_j)
SL	positive slack: $max\{d_j - p_{ij} - time, 0\}$

different meta-algorithms. Such a modular design makes it possible to use the same meta-algorithm for different scheduling criteria, only by using the appropriate priority function.

Algorithm 1 represents the meta-algorithm which we use for performing scheduling in the unrelated machines environment for an arbitrary criterion. Unlike the priority functions, the meta-algorithm is not evolved, but defined manually. This algorithm finds an appropriate mapping between a job and a machine based on the value of the priority function (which is evolved with GP). If the most suited machine for the job is available, then the job is scheduled for execution on that concrete machine. Otherwise, the scheduling is postponed until a machine becomes available, or a new job becomes ready.

The complexity of this algorithm is comparable to the complexity of existing constructive scheduling heuristics, which makes it possible for this approach to be used in on-line scheduling [27]. Since the complexity of an evolved priority function is always $O(1)$, the described meta-algorithm finds the best priority value in $O(m * n)$ complexity, where n denotes the total number of available jobs, and m the total number of machines.

A complete time complexity of this approach may be divided in two parts: the first part is the evolution of the priority function, and the second part is its application in real-time conditions. Although the evolution of the priority function using genetic programming is a very demanding procedure, with a complexity comparable to other search-based methods, it can be performed at any time before the priority function is actually used. This is true for any existing scheduling heuristic, which are normally designed before the actual application. In that way, multiple priority functions for certain optimization criteria can be prepared beforehand and then simply injected into the meta-algorithm.

The second part represents the complexity of the meta-algorithm which uses the evolved function, and this is negligible compared to any search-based algorithm - the schedules for 120 problem instances used in this paper can be generated in less than a second, as well as with other constructive heuristics.

Algorithm 1 Meta-algorithm used for GP scheduling

```
1: while unscheduled jobs are available do
2:     wait until a job becomes ready or a job finishes;
3:     for all available jobs and all machines do
4:         obtain the priority π_ij of job j on machine i;
5:     end for
6:     for all available jobs do
7:         determine the best machine (the one for which
8:         the best value of priority π_ij is achieved);
9:     end for
10:    while jobs whose best machine is available exist
11:     do
12:        determine the best priority of all such jobs;
13:        schedule the job with the best priority;
14:    end while
15: end while
```

4. Optimizations

In order to achieve the best results with the aforementioned scheduling procedure, several optimization procedures were tried out.

4.1. GP Algorithm Parameters

Table 2 represents the parameters of the GP. The values of all GP parameters were optimized in order to minimize the generalization error of the algorithm [28]. The set of the crossover and mutation operators was also optimized, as it was shown that by removing some of the genetic operators better fitness of the evolved solutions can be achieved. All of the resulting crossover and mutation operators listed in the table are applied with the same probability, meaning that on the average all operators will be used the same number of times.

The optimal set of genetic operators was determined by using both simple constructive and destructive heuristics. Whereas the constructive heuristic starts with an empty set of elements, sequentially adding elements which happen to increase the quality of evolved solutions, the destructive one starts with a set containing all elements and removes them one by one if they happen to increase the quality of evolved solutions. Both of these heuristics were used to determine the "optimal" set of genetic operators. The better one of those two "optimal" sets of genetic operators was chosen and used in the experiments. In the following subsections the described procedure will be used to find optimal sets of other elements as well.

The initial parameter values were chosen as a rule of thumb and were used as a starting point for optimisations which resulted in the parameters presented in the table. The values of the initial parameters are mostly the same as the ones from Table 2, except for the maximal tree depth which was initially 7 and the crossover operator set which included the one-point crossover operator. It should be noted that the GP parameters were tuned by optimizing the weighted tardiness criterion, after which the resulting parameters were used for evolving priority functions for other criteria.

4.2. Selection of Functional Nodes

Table 3 represents the set of available functional nodes we experimented with. The $IFGT$ and $IFLT$ nodes (simple if statements) were added in order to allow for different parts of the evolved expressions to specialize for certain cases. The remaining nodes represent simple mathematical operators, of which the protected division is used in order to prevent division by zero. The square root operator was implemented as a safe operator (similar to the division operator) to prevent illegal values from appearing. It would probably

Table 2: Parameters for the GP

Parameter	**Value**
population size	1000
stop criteria	maximum number of iterations (80 000)
selection	steady state GP using tournament selection
tournament size	3
initialization	*ramped half-and-half*
mutation probability	0.3
maximal tree depth	5
crossover operators	subtree, uniform, context-preserving, size-fair
mutation operators	subtree, Gauss, hoist, node complement, node replacement, permutation, shrink

prove better if a more sophisticated approach was used in order to handle such cases (e.g. by using interval arithmetic [29]), but this is left for future research.

The initial subset of functions used in constructive feature selection consisted only of four arithmetic operators, $+, -, *$ and protected division. By using constructive and destructive heuristics, an optimal set of the functional operators was found which comprises of four basic mathematical operators $(+, -, *, /)$ and the unary operator POS.

4.3. Dimensionally Aware Genetic Programming

The solutions generated by genetic programming are generally not *semantically correct*, which holds for all GP applications [30]. However, it is possible to restrict the GP search process only to semantically correct solutions. This approach is known as dimensionally aware genetic programming (DAGP) [31].

Dimensionally aware genetic programming is a variant of genetic programming in which solutions of the algorithm represent semantically correct expressions. This means that there are constraints imposed on the functional nodes in order to regulate which elements can be their children. For example, a simple semantic rule would be that the addition operator can be performed only on the nodes whose values are in the same unit (e.g. seconds). In such a way it is possible to evolve priority functions which are more similar to the ones which would be devised by human experts.

This certainly increases the readability of the generated solutions; on the other hand, it introduces added complexity into the genetic programming. The individuals can no longer be generated completely by random, but an algorithm is needed in order to generate solutions which are semantically correct. Furthermore, crossover and mutation operators need to be adjusted as well, in order to preserve the semantic correctness of individuals.

In order to ensure the semantic correctness of the solutions, it is required that all nodes carry the information about the physical unit (e.g. seconds, meters), as well as the exponent to which the unit is raised (e.g. s^2, s^{-1}). Since all terminal nodes in the this application have the same unit - the time (e.g. in seconds), it is sufficient to carry only the information about the unit exponent, since the unit will be the same for all nodes. The only exception is the weight node, which in itself has no unit, but this can be treated as if the unit is raised to the zero exponent. Table 4 lists the semantic rules for the functional nodes applied in our implementation of dimensionally aware GP. As a consequence, this will prevent the addition

Table 3: Functional nodes

Node name	**Description**
+	binary addition operator
-	binary subtraction operator
*	binary multiplication operator
/	secure binary division: $/(a,b) = \begin{cases} 1, \text{ if } \lvert b \rvert < 0.000001 \\ \frac{a}{b}, \text{else} \end{cases}$
POS	unary operator: $POS(a) = max\{a, 0\}$
IFGT	$IFGT(a,b,c,d) = \begin{cases} c, \text{ if } a > b \\ d, \text{ else} \end{cases}$
IFLT	$IFLT(a,b,c,d) = \begin{cases} c, \text{ if } a < b \\ d, \text{ else} \end{cases}$
MAX	$MAX(a,b) = \begin{cases} a, \text{ if } a > b \\ b, \text{ else} \end{cases}$
MIN	$MIN(a,b) = \begin{cases} a, \text{ if } a < b \\ b, \text{ else} \end{cases}$
SQRT	$SQRT(a) = \begin{cases} 1, \text{ if } a < 0 \\ \sqrt{a}, \text{ else} \end{cases}$
AVG	$AVG(a,b) = (a+b)/2$
ABS	$ABS(a) = \lvert a \rvert$

of weight and processing time of a job (for example), or any other combination of values with non-matching time exponents.

Figure 1 represents two sample solutions. The left solution is semantically incorrect because it tries to subtract two nodes which have different unit exponents: $w - pt$. On the other hand, the right expression represents a semantically valid expression because it adheres to all the aforementioned semantic rules.

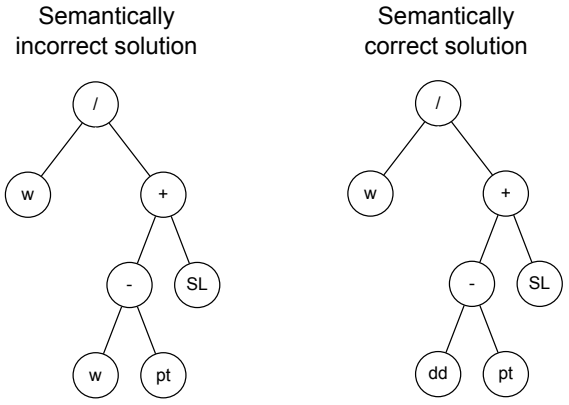

Figure 1: Example of a semantically incorrect and a correct solution

Table 4: The defined semantic rules

Node	**Constraint**	**New exponent value**
+	the left and right child must have the same exponent	the same as the exponent of the child nodes
-	the left and right child must have the same exponent	the same as the exponent of the child nodes
*	none	the sum of the exponents of the left and right child
/	none	the difference between the exponents of the left and right child
pos	none	the same as the exponent of the child

Our experiments (shown in section 5) indicate that there is no significant difference between the results obtained by these two approaches. However, this suggests that the DAGP can be used to generate semantically sound solutions without a loss in solution quality.

4.4. Gene Expression Programming

Gene expression programming (GEP) is an evolutionary algorithm similar to Genetic programming [32]. The only real difference between those two approaches is how the solutions are represented. GEP individuals are always of the same size even though all nodes may not be used during the construction of the expression tree. The idea behind this design is to simplify the genetic operators which now do not need to operate on a tree structure, but rather on an array. GEP was used for evolving scheduling rules in the single machine environment and has shown promising results [33].

A GEP individual consists of several genes where each gene comprises the same number of nodes. Additionally, each gene is divided in two parts: the head and the tail of the gene. The head can contain both terminal and functional nodes, while the tail can contain only terminal nodes. While the head size is often presented as a parameter of the algorithm, the tail size is calculated as $t = h * (n_{max} - 1) + 1$, where h represents the head size and n_{max} the maximum number of children per function node.

A sample GEP individual is presented in Figure 2. As mentioned before, each individual consists of several genes which are separated by the "|" delimiter. The underlined part of the gene represents a *K-expression*, a part of the gene used to create an expression tree. The rest of the gene is not used and it is here only to preserve the constant gene size.

If GEP individuals consist of more than one gene, then a *linking function* must be defined which determines how the individual gene expressions are combined. These linking functions are a part of the individuals as well and are evolved through the algorithm, giving more freedom for the evolved priority functions to adapt to the scheduling conditions. Figure 2 shows a GEP individual, where the linking nodes are located at the very beginning. In our implementation the linking nodes are always defined as functions with two operands.

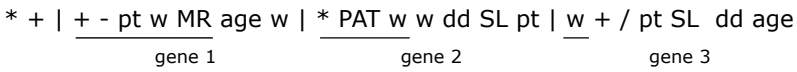

Figure 2: Example of a GEP individual

Expression tree generated from this sample individual is presented in Figure 3. Each gene's nodes are represented in a different color (in this case, in blue, green and purple). The red nodes denote functional nodes which are used to combine genes with each other. The procedure of transforming the individual into the expression tree is described in [34].

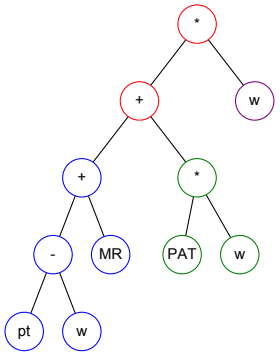

Figure 3: Example of an expression tree built from a GEP individual

Regarding the implementation, we used one crossover operator (one point crossover), one mutation operator (replacement mutation) and three transposition operators (IS, RIS and gene transposition). Two GEP parameters were optimized, the head size and the number of genes. It was found that GEP obtains the best results when the individual consists of three genes and when the gene head contains six nodes.

4.5. Scheduling with Iterative Dispatching Rules

Producing GP heuristics with iterative dispatching rules (IDR) was first proposed in [35], where the approach was applied to the job shop scheduling problem. The core idea of this approach is to build a better schedule throughout multiple iterations (hence the name) using a special kind of priority function. The used function is interesting in the fact that it does not depend only on the general properties of machines and jobs, but also on information extracted from the previous iterations, i.e. from the previously built schedules. The intuition behind this kind of approach is that this way the priority function can collect much more information about the schedule as a whole, in contrast to the traditional priority functions which use only the information from the partially generated schedules. In other words, this approach considers the future impacts of the decisions made by the priority function as well.

The algorithm works as follows: the first time the schedule is built, the properties of the previously built schedules are set to some predefined values. However, when the schedule is built for the second time, the priority function uses the properties of the previously created schedule. If the resulting schedule is "better" than the old one, the process is repeated, otherwise, the process is stopped and the previously built schedule is returned as a result. This approach can be used with scheduling rules devised by any of the previous methods.

In order to accommodate for the information about the previous schedule, new tree nodes need to be included. This additional node set comprises of three terminal nodes and one functional node. Their descriptions are given in Table 5.

$NLATE$ is a node which represents the number of late jobs in the previous schedule. When the schedule is built for the first time (i.e. there is no information about the previously built schedules), the value of this terminal node is equal to the total number of jobs (presuming that all jobs are late). Furthermore, $INDLATE$ terminal node represents the lateness of a specific job in the previous schedule. Lateness of a

job with the index j is defined as: $L_j = C_j - d_j$. The last terminal node $LATENESS$ is defined as total lateness of the previously generated schedule. The default values of these two nodes are set to large values (ones which can not be achieved by any feasible schedule). The only functional node is the $ISLATE$ node which determines if the current job was late in the previous schedule in which case it executes its first child. Otherwise, it executes the second child.

Table 5: Nodes introduced for the GP with IDR

Node name	**Description**
NLATE	the number of late jobs
LATENESS	the lateness of all the jobs
INDLATE	the individual lateness of a job
ISLATE	if the job was late executes the left child, else it executes the right child

It should be noted that this set of nodes was designed with the weighted tardiness criteria in mind. Nevertheless, these nodes can be used in the optimization of other criteria as well. However, better results could possibly be achieved by selecting unique nodes for each criterion separately.

The optimal set of the aforementioned nodes was found using the constructive and destructive heuristic. The best average solution was achieved by using only the $NLATE$ node. On the other hand, the single best overall solution (for the weighted tardiness criteria) was found by the GP which uses only the $LATENESS$ node. Other combinations of nodes did not produce significantly better results.

It is important to note that the IDR approach imposes a major constraint: it cannot be used in on-line conditions when no information about future jobs is available, because a complete schedule cannot be built. This may lead to the conclusion that IDRs may only be used in static (off-line) scheduling. However, they can still be used in *dynamic conditions*, when the system parameters may change during the run, such as the arrival of new jobs or machine breakdowns. In these conditions the search-based methods are generally not applicable, because they cannot react quickly to changes, whereas the time complexity of IDRs is still negligible and comparable to existing heuristics.

5. Benchmarks and Results

5.1. Benchmark Setup

To gain a realistic insight into the performance of the developed system, an extensive set of benchmarks was performed. A set of 120 problem instances was generated based on the methods described in related references for various scheduling environments [5, 6, 36, 37].

The set of 120 problem instances was divided into two disjoint sets, the training set and the validation set. Each of these sets contained 60 problem instances. Depending on the problem instance, the total number of jobs can be 12, 25, 50, or 100, while the total number of machines is 3, 6 or 10. The number of test instances is the same for each combination of the number of jobs and machines (i.e. there are 10 instances for each of the 12 combinations of these parameter values). The values of processing times of the jobs are generated in the interval $p_{ij} \in [0, 100]$ using one of the following three probabilistic distributions: uniform, normal (Gaussian) and quasi-bimodal. The choice from which of these three distributions the processing times will be drawn is chosen randomly for each job (with each distribution having the same probability of being chosen). The job weights follow a uniform distribution in $[0, 1]$. The release times of the jobs were generated, using a uniform distribution, from the interval

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

where $\hat{p}$ is defined as

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

and p_{ij} denotes the duration of job j on machine i while m denotes the total number of machines. The due dates of the jobs were also defined using a uniform distribution from the interval

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

In this equation, parameter T represents the due date tightness, while the parameter R represents the due date range. Both of those parameters assumed values of 0.2, 0.4, 0.6, 0.8 and 1 in various combinations while generating the problem set. Additional details about data generation and the problem instances are available at the project web site at [38].

The *training set* of problem instances was used by the GP in order to evolve the priority functions by optimizing four scheduling criteria: weighted tardiness, weighted number of tardy jobs, flow-time and makespan. Needless to say, each of these criteria was optimized independently. Following this step, the effectiveness of the generated priority functions was evaluated using the *validation set* and results on this set are presented in the following sections.

Since the problem instances have different characteristics (number of jobs, machines, durations), we adjusted the objective functions in order for all problem instances to have an equal effect on the algorithm. Therefore, the *normalized* objective functions for the problem instance with the index i are defined as follows:

- for the weighted tardiness criterion $f_i = \frac{\sum_{j=1}^{n} w_j T_j}{n\bar{w}\bar{p}}$
- for the weighted number of tardy jobs criterion $f_i = \frac{\sum_{j=1}^{n} w_j U_j}{n\bar{w}}$
- for the flowtime criterion $f_i = \frac{\sum_{j=1}^{n} F_j}{n\bar{p}}$
- for the makespan criterion $f_i = \frac{\max\{C_j\}}{n\bar{p}}$,

where n denotes the number of jobs in the problem instance, $\bar{w}$ the average weight of the jobs and $\bar{p}$ the average job processing duration. The total objective function is then calculated as the sum of the objective functions of the individual problem instances.

In order for the benchmark results to be statistically significant, each experiment was executed 50 times, while preserving the best solution from each run. The resulting best solutions from these runs were then used to calculate quantitative information such as the mean fitness value of the solutions, the median of the fitness, the minimum and the maximum fitness value, and the standard deviation.

5.2. Benchmark Algorithms

To assess the quality of schedules obtained by using the evolved priority functions, the obtained results were compared to the results of four constructive heuristic scheduling methods. The methods used for comparison are: min-min [39], max-min [40], sufferage[41], min-max [42]. These methods were used to generate schedules for the problem instances in the validation set.

Search-based methods were also used in order to generate schedules for the problem instances in the validation set. These methods were employed primarily to estimate a reasonable lower bound to the problem instances, since no known instances with all the properties and lower bounds are available. The methods covered in this project include several variants of genetic algorithm, ant colony optimization and hybrid evolutionary algorithms with several local search operators. In these meta-heuristics we used two encodings: the permutation encoding, which defines the relative order of the jobs, and the floating point encoding, which decodes the floating point values into job priorities (a priority vector). The ACO algorithm is used with permutation encoding, while the GA is used both with permutation and priority vectors. We experimented

with several selection schemes for GA (steady state with tournament selection, generation with roulette-wheel) and several genetic operators, as well as a combination with deterministic local search operator with pattern search. Based on our experiments, the floating point encoding provided better results.

The execution times of these algorithms vary greatly depending on the values of the parameters (population size, number of iterations). Producing schedules for 60 problem instances lasted from a few hours up to a whole day. For comparison, generating schedules for the same problem instances using constructive scheduling heuristics took less than a second. Furthermore, all the search-based algorithms were executed at least 30 times in order to obtain the best possible results.

It is important to note that only the overall single best result for each problem instance found by any of these methods was used for comparison. This means that the total result denoted as *"search-based"* is actually a combination of best results from several meta-heuristic algorithms and does not represent a result achieved by any single algorithm. It should be also noted that these approaches can only perform off-line scheduling; consequently they are expected to obtain better results than the other approaches which also perform on-line scheduling. We omit further implementation details since these methods cannot be used in dynamic scheduling conditions and are not the focus of this work.

5.3. Result Comparison - On-line Scheduling

In this subsection we present the results of GP variants that can be used in on-line scheduling (without the iterative GP). This includes four variations: standard GP without parameter optimisations, standard GP with parameter optimisations, dimensionally aware GP and GEP. The GP without parameter optimisation was included in order to asses the influence which parameter optimisation can have on the achieved results.

Table 6 shows the complete results for all criteria and all the described approaches. In the following subsections the approaches are compared with each other based on the four aforementioned scheduling criteria. For each criterion the results for the GP approaches are presented in a *Tukey box plot* [43]. The red stars in the box plot diagrams denote the maximum outlier (if it exists) for a certain experiment. Aside from the box plots we have also performed statistical tests between all the variants using the Wilcoxon rank-sum statistical test. The differences are considered statistically significant if the obtained value for p is smaller than 0.05.

5.3.1. Weighted Tardiness Criterion

In this section the results for the weighted tardiness (Twt) criterion will be compared. From table 6 it can be seen that the constructive scheduling heuristics achieve quite similar results for this criteria (except from the max-min heuristic), with the sufferage heuristic achieving the best result.

The different GP methods have shown more or less similar results as can be seen from figure 4 which shows results in the *box plot* representation. The GP method without any parameter optimisations obtained the worst overall results from all the GP methods and it can be seen that the GP with the optimised parameters can achieve better results with a much smaller standard deviation. The statistical tests show that GP without parameter optimisations is worse than the other approaches. When compared to the standard GP with optimised parameters, dimensionally aware GP and GEP achieved results which are to a small extent worse, but the statistical tests show that there is no significant difference between those three methods.

From table 6 it can be also seen that all the GP methods outperform the existing scheduling heuristics. Even the worst solutions found by the GP methods are still better than the solutions found by the scheduling heuristics (except for GP without parameter optimisations). The search-based methods, as expected, provided the best combined result.

5.3.2. Weighted Number of Tardy Jobs Criterion

In this section the results for the weighted number of tardy jobs (Nwt) criterion will be compared. From the set of the constructive heuristics, the min-min heuristic achieved the best result, with the sufferage offering also quite good results. The other two heuristics fared considerably worse.

Figure 5 represents the *box plot* representation of the results achieved for this criterion. It can be seen that GP without optimisations achieves results more or less comparable to the GP with optimised parameters.

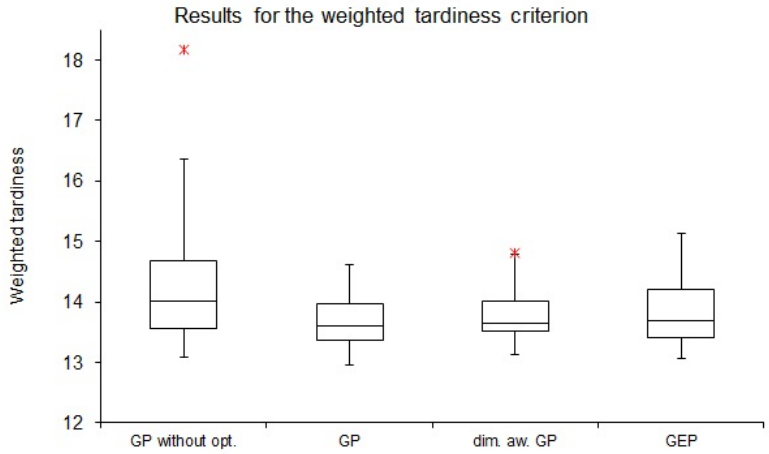

Figure 4: *Box plot* for the weighted tardiness criterion

This is backed up with statistical tests which show no significant difference between those approaches. Dimensionally aware GP and GEP provide a better average of the best solutions, when compared to the standard GP with and without optimisations, but both of these were unable to find a better solution than the best solution found by the standard GP. Statistical tests showed that there is a significant difference between GP without optimisations and dimensionally aware GP (p-value=0.033) and also between GP without optimisation and GEP (p-value=0.044). On the other hand, there is no significant difference between GP with parameter optimisations and dimensionally aware GP or GEP.

It should also be noted that all the tried out GP variants have a smaller standard deviation of the best solutions when compared the standard GP (with and without parameter optimisations). This means the best found solutions are less scattered, which in turn means that we have a greater probability of finding a good solution than when using the standard GP.

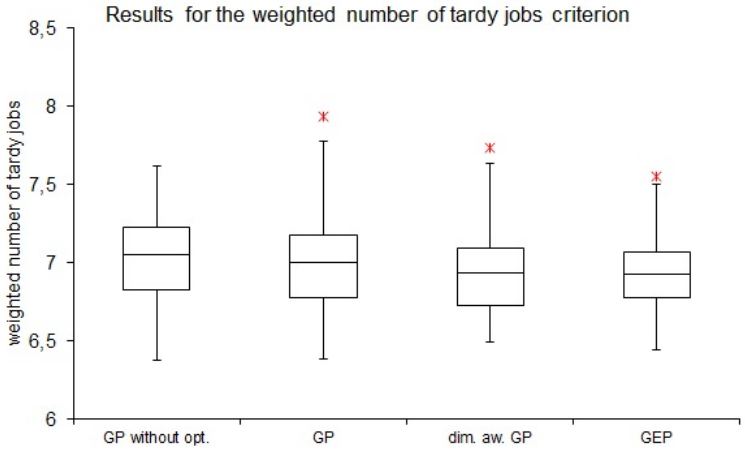

Figure 5: *Box plot* for the weighted number of tardy jobs criterion

When the results of the GP variants are compared to the heuristic scheduling methods, it can be seen that GP delivers better results in general. Once again the search-based methods provided the result which was by far better when compared to the rest of the algorithms.

Another thing that can be deduced from table 6 is that generally better results for the Nwt criterion

are achieved when the priority function is evolved for the weighted tardiness. This means that it is more worthwhile to optimize the Twt criterion, because at the same time the Nwt criterion is also optimized.

5.3.3. Flowtime Criterion

In this section the results for the flowtime (Ft) will be compared. As can be seen from table 6, the min-min heuristic achieved the best result among the heuristic scheduling methods, followed by the sufferage heuristic.

The *box plot* representation of the results achieved for the flowtime criterion are shown in figure 6. For this objective the GP approaches have shown some interesting results. Although GP with parameter optimisations was able to achieve a better average of the best solutions when compared to GP without optimisations, the overall best result was worse then the one found by GP without optimisations. Although dimensionally aware GP managed to find the best overall result, the statistical tests show that there are no significant differences between dimensionally aware GP and the two standard GP approaches. On the other hand, tests show that the results achieved by GEP are significantly different than those achieved of any of the three former approaches.

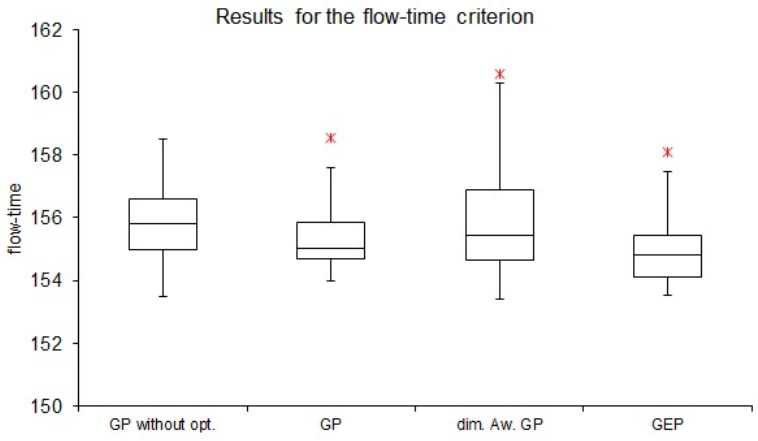

Figure 6: *Box plot* for the flowtime criterion

Even for this criteria it is shown that the GP approaches outperform the heuristic scheduling methods. Only the min-min heuristic achieved results which are close to the solutions achieved by the GP approaches.

5.3.4. Makespan Criterion

In this section, the results for the last, the makespan criterion, will be presented. It should be noted that this objective's validity is more justified in static scheduling, since in on-line conditions the jobs that arrive the last determine the overall makespan. However, we include this objective for completeness and comparison purposes.

Among the constructive methods, the sufferage heuristic achieved the best result. The min-max heuristic obtained the second best result and the max-min heuristic came last for this criteria as well.

Figure 7 represents the *box plot* representation of the achieved results of the GP approaches. From the results for this criterion it can be seen that GP with optimisations, dimensionally aware GP and GEP achieve quite similar results. GP without optimisations managed to obtain the overall best result when compared to the previous approaches, but the average value and the standard deviation was worse. Nevertheless, the statistical tests show that significant difference exists only between GP without optimisations and GEP, but not between GP without optimisations and other approaches.

For this criteria the differences between the GP approaches and the heuristic scheduling methods are very small. The scheduling heuristics have shown to provide results which are competitive to the ones achieved by

the GP approaches (which is expected given the on-line scheduling conditions). The search-based methods were able to find the best overall solution for this criterion, thus providing a reasonable lower bound.

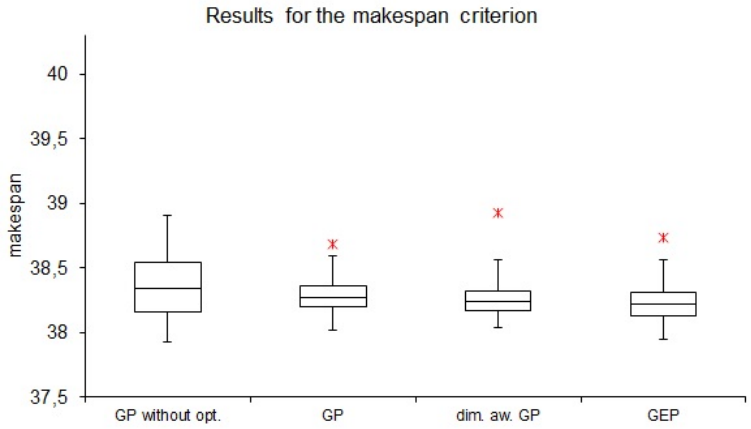

Figure 7: *Box plot* for the makespan criterion

5.4. Result Comparison - Off-line Scheduling

In this section we present the results with iterative dispatching rules (IDRs); these rules use terminal nodes with information that is only available at the end of the schedule. Using the information from the previously built complete schedules, the scheduler can update those terminal values and attempt to improve schedule quality. In this paper we experimented with several additional terminals, but experiments indicated that using only the node with number of late jobs ($NLATE$) obtained the best average result. The iterative scheduling was applied to the GP with optimized parameter values for all the criteria and the results are given in Fig. 8 and Table 6.

It is clear that the iterative rules provide the advantage over all the previous methods which build the schedule in a single pass (the statistical tests show significant difference with $p < 0.001$ when compared to previous methods). However, this mechanism can only be used when all the information about the future jobs is available, which corresponds to static scheduling. In these conditions, the iterative approach should be comparable to search-based methods, but the performance of the search-based methods is still much better.

It is possible that, with the inclusion of other terminals that provide more relevant information about the previously built schedules and current criterion, better results could be obtained, but this is left for future research, since the main focus of this paper are the on-line scheduling conditions.

6. Discussion

From the results presented in the last section a great number of conclusions can be drawn. First of all, we will shortly assess the performance of the constructive heuristic scheduling methods. Among the tried out methods, the min-min and sufferage proved to be the most promising. Both of these methods found the best solution (among the heuristic scheduling approaches) for two criteria, min-min for the Nwt and Ft criteria and sufferage for the Twt and Cmax criteria. The max-min heuristic has proven to be the worst among these, achieving the worst results for all criteria. The min-max heuristic, on the other hand, has shown mixed results, providing a good result only for the Cmax criteria. Naturally there are many other scheduling heuristics and even modifications of the aforementioned ones which could possibly provide better results than the four selected ones [44, 45, 46, 47, 48].

Among the GP approaches, GP with IDR has proven to be the best approach, which was expected with the additional information from multiple iterations at its disposal. However, this information can only be

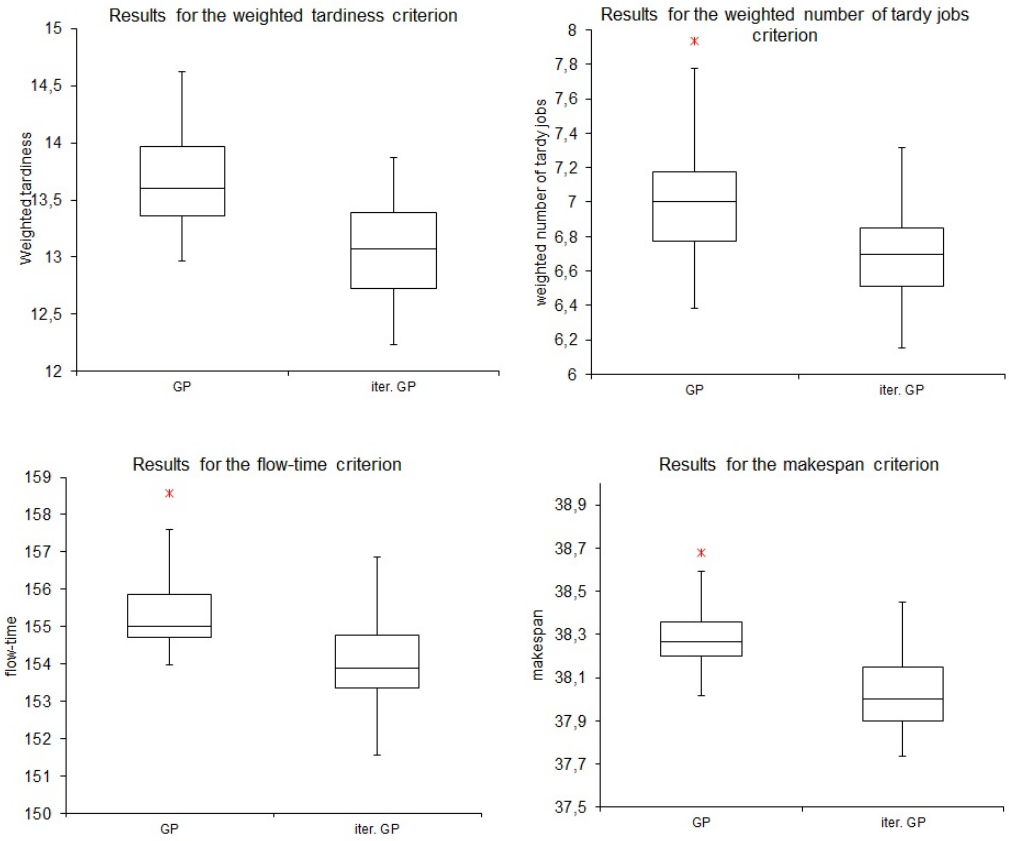

Figure 8: *Box plot* images for all the criteria - iterative dispatching rules

exploited if data about future jobs is available. The other GP methods provided more or less similar results, with GEP and dimensionally aware GP even providing slightly better solutions for some criteria when compared to the standard GP. Additionally, the dimensionally aware GP has the advantage of providing "semantically" correct solutions, while on the other hand GEP has generally evolved priority functions which are simpler when compared to the priority functions of the other approaches. A sample solution obtained by the use of dimensionally aware GP for minimizing the Twt criterion is

$$\begin{aligned}\pi = pos(pos((w * SL))/((w * pavg)/(SL + pmin)))- \\ (pos((w * pt)) + (w * age) - ((pmin/w)- \\ (MR - age) + (dd - pt) + (pmin + pavg))).\end{aligned}$$

This solution achieves a value of 13.8299 for the above criterion. Upon closer examination it can be seen that the solution is semantically correct and conforms to all the rules stated in section 4.3.

The tests also show that optimising the GP parameters can have an influence on the results. For the weighted tardiness criterion it was shown that there is a substantial difference between the 'standard' GP and optimised GP. Although there is no great difference between the best found solutions of those two approaches, GP with optimised parameters achieved a better average of the best results from all runs and also had a much smaller standard deviation. Based on these observations we can conclude that optimising the parameters did not really result in an approach which was able to find better results, but rather in an approach which is more robust and is more likely to find good results than the approach without any parameter optimisations. For the other criteria the difference between those two approaches is not as prominent, which is probably due to the fact that the the parameters were optimised only for the weighted tardiness criterion and then used for all the other criteria.

When comparing the GP approaches with the heuristic scheduling methods it can be seen that they clearly outperform them in all criteria except for the Cmax criterion. A great advantage of existing heuristics is that they perform reasonably well over multiple criteria, while the GP approaches tend to perform well only for the criteria for which the priority function was devised (the only exception being the Nwt quality when the Twt criterion is optimized). Based on this observation, it can be concluded that if more than one criterion should be optimized at once, it is more practical to use the existing heuristic scheduling methods. On the other hand, if it is important to optimize a single criteria as much as possible, or if the objective is based on user preferences, the GP approaches provide a better option.

Even though the GP-based approaches can be used in off-line scheduling as well, they do not provide solutions which are as good as the ones provided by search-based procedures, which was seen in the previous section. If there is no limit imposed on the time needed to build a schedule, search-based metaheuristics should be the methods of choice. Otherwise, if time is of the essence, one of the presented approaches should be considered rather than the search-based methods.

When all things are considered, it can be concluded that there is no single approach which provides near optimal solutions for all criteria and environments. Search-based methods are suited for off-line scheduling, but are not applicable to scheduling in the dynamic environment. Existing heuristics have shown to provide a good overall performance and even solutions which are competitive with those of the GP approaches for the Cmax criterion, making them likely to be a better choice for optimizing that criterion. GP approaches have however proven to be able to achieve very good results for any criteria for which the priority function was evolved. When compared to the search-based methods, the GP approaches offer speed which is comparable to the speed of existing scheduling heuristics (if the priority functions are prepared beforehand).

More importantly, and similarly to search-based methods, GP can be optimized for almost any conceivable user criteria, while the existing scheduling heuristics do not provide such a possibility (it would be required to design a new heuristic for optimizing a specific criterion). Considering all these aspects, choosing the right approach for optimizing a given criterion depends on many factors and end user requirements.

7. Conclusion

This paper shows how genetic programming can be used to build scheduling algorithms for the parallel unrelated machines scheduling environment with arbitrary scheduling criteria. The proposed heuristic is composed of two parts: a meta-algorithm and a priority function. The meta-algorithm we propose is defined manually, while the priority function is evolved using GP. This allows the users to specify an arbitrary criterion, and evolve the appropriate priority function for it.

The experiments have shown that the proposed algorithm achieved results which were in most cases better than the results achieved by the existing scheduling heuristics. The GP was still unable to find solutions better than those found by the search-based methods. However, the goal of this approach is not to provide optimal or near optimal solutions, but to find solutions with acceptable quality in a small amount of time.

Additionally, several different GP approaches like dimensionally aware GP, GEP and GP with iterative dispatching rules were tried out. GP with iterative dispatching rules achieved the best results when compared to any of the other GP approaches, but is applicable only in off-line scheduling. Dimensionally aware GP and GEP achieved results which were mostly comparable to the standard GP, but offer some benefits which could make them more appropriate for certain situations.

In future work we plan to further investigate the optimizations for the GP in order to achieve even better results and also to try out combinations of the aforementioned modifications. While the evolution of priority functions is central to this and existing research, the evolution of the meta-algorithm itself, rather than designing it manually, is also a challenging task which has not received much attention so far. Additionally, it is also planned to adjust the GP, by adding new terminal nodes, in order for it to be more viable for the off-line scheduling conditions.

References

[1] M. L. Pinedo, Scheduling: theory, algorithms, and systems, Springer Science & Business Media, 2012.

[2] A. Jones, L. Rabelo, Survey of job shop scheduling techniques, tech. rep., NISTIR, National Institute of Standards and Technology.

[3] J. Branke, C. W. Pickardt, Evolutionary search for difficult problem instances to support the design of job shop dispatching rules, European Journal of Operational Research 212 (1) (2011) 22–32.

[4] T. P. Adams, Creation of simple, deadline, and priority scheduling algorithms using genetic programming, Genetic Algorithms and Genetic Programming at Stanford 2002 (2002) 84.

[5] C. Dimopoulos, A. M. Zalzala, Investigating the use of genetic programming for a classic one-machine scheduling problem, Advances in Engineering Software 32 (6) (2001) 489–498.

[6] C. Dimopoulos, A. M. Zalzala, A genetic programming heuristic for the one-machine total tardiness problem, in: Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on, Vol. 3, IEEE, 1999.

[7] D. Jakobović, L. Budin, Dynamic scheduling with genetic programming, in: Genetic Programming, Springer, 2006, pp. 73–84.

[8] D. Jakobović, K. Marasović, Evolving priority scheduling heuristics with genetic programming, Applied Soft Computing 12 (9) (2012) 2781–2789.

[9] K. Miyashita, Job-shop scheduling with gp, in: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000), 2000, pp. 505–512.

[10] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem, Evolutionary Computation, IEEE Transactions on 17 (5) (2013) 621–639.

[11] R. Hunt, M. Johnston, M. Zhang, Evolving less-myopic scheduling rules for dynamic job shop scheduling with genetic programming, in: Proceedings of the 2014 conference on Genetic and evolutionary computation, ACM, 2014, pp. 927–934.

[12] T. Hildebrandt, J. Heger, B. Scholz-Reiter, Towards improved dispatching rules for complex shop floor scenarios: a genetic programming approach, in: Proceedings of the 12th annual conference on Genetic and evolutionary computation, ACM, 2010, pp. 257–264.

[13] N. B. Ho, J. C. Tay, Evolving dispatching rules for solving the flexible job-shop problem, in: Evolutionary Computation, 2005. The 2005 IEEE Congress on, Vol. 3, IEEE, 2005, pp. 2848–2855.

[14] D. Jakobović, L. Jelenković, L. Budin, Genetic programming heuristics for multiple machine scheduling, in: Genetic Programming, Springer, 2007, pp. 321–330.

[15] V. Cheng, L. Crawford, P. Menon, Air traffic control using genetic search techniques, in: Control Applications, 1999. Proceedings of the 1999 IEEE International Conference on, Vol. 1, IEEE, 1999, pp. 249–254.

[16] J. V. Hansen, Genetic search methods in air traffic control, Computers & Operations Research 31 (3) (2004) 445–459.

[17] C. W. Pickardt, T. Hildebrandt, J. Branke, J. Heger, B. Scholz-Reiter, Evolutionary generation of dispatching rule sets for complex dynamic scheduling problems, International Journal of Production Economics 145 (1) (2013) 67–77.
[18] J. Branke, S. Nguyen, C. Pickardt, M. Zhang, Automated design of production scheduling heuristics: A review, Transactions on Evolutionary Computation.
[19] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem., IEEE Trans. Evolutionary Computation 17 (5) (2013) 621–639.
[20] J. Branke, T. Hildebrandt, B. Scholz-Reiter, Hyper-heuristic evolution of dispatching rules: A comparison of rule representations, Evol. Comput. 23 (2) (2015) 249–277.
[21] R. Poli, W. B. Langdon, N. F. McPhee, J. R. Koza, A field guide to genetic programming, Lulu. com, 2008.
[22] P. G. Espejo, S. Ventura, F. Herrera, A survey on the application of genetic programming to classification, Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on 40 (2) (2010) 121–144.
[23] H. Jabeen, A. R. Baig, Review of classification using genetic programming, International journal of engineering science and technology 2 (2) (2010) 94–103.
[24] I. Icke, J. C. Bongard, Improving genetic programming based symbolic regression using deterministic machine learning, in: Evolutionary Computation (CEC), 2013 IEEE Congress on, IEEE, 2013, pp. 1763–1770.
[25] J. R. Koza, Human-competitive results produced by genetic programming, Genetic Programming and Evolvable Machines 11 (3-4) (2010) 251–284.
[26] D. Jakobović, Evolutionary computation framework.
URL `http://gp.zemris.fer.hr/ecf`
[27] M. Pinedo, Offline deterministic scheduling, stochastic scheduling, and online deterministic scheduling: A comparative overview, Handbook of Scheduling. Chapman & Hall/CRC.
[28] I. Kushchu, Genetic programming and evolutionary generalization, Evolutionary Computation, IEEE Transactions on 6 (5) (2002) 431–442.
[29] M. Keijzer, Improving symbolic regression with interval arithmetic and linear scaling, in: Genetic programming, Springer, 2003, pp. 70–82.
[30] J. R. Koza, Genetic programming: on the programming of computers by means of natural selection, Vol. 1, MIT press, 1992.
[31] M. Keijzer, V. Babovic, Dimensionally aware genetic programming, in: Proceedings of the Genetic and Evolutionary computation Conference, Vol. 2, 1999, pp. 1069–1076.
[32] C. Ferreira, Gene expression programming: A new adaptive algorithm for solving problems, Complex Systems 13 (2001) 87–129.
[33] L. Nie, X. Shao, L. Gao, W. Li, Evolving scheduling rules with gene expression programming for dynamic single-machine scheduling problems, The International Journal of Advanced Manufacturing Technology 50 (5-8) (2010) 729–747.
[34] X. Li, C. Zhou, W. Xiao, P. C. Nelson, Prefix gene expression programming, in: Proc. Genetic and Evolutionary Computation Conference, Washington, 2005, pp. 25–31.
[35] S. Nguyen, M. Zhang, M. Johnston, K. C. Tan, Learning iterative dispatching rules for job shop scheduling with genetic programming, The International Journal of Advanced Manufacturing Technology 67 (1-4) (2013) 85–100.
[36] Y. H. Lee, K. Bhaskaran, M. Pinedo, A heuristic to minimize the total weighted tardiness with sequence-dependent setups, IIE transactions 29 (1) (1997) 45–52.
[37] M. Pfund, J. W. Fowler, A. Gadkari, Y. Chen, Scheduling jobs on parallel machines with setup times and ready times, Computers & Industrial Engineering 54 (4) (2008) 764–782.
[38] D. Jakobović, Project site.
URL `http://gp.zemris.fer.hr/scheduling/`
[39] E. Davis, J. M. Jaffe, Algorithms for scheduling tasks on unrelated processors, Journal of the ACM (JACM) 28 (4) (1981) 721–736.
[40] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hensgen, et al., 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 (6) (2001) 810–837.
[41] D. Hensgen, R. F. Freund, Dynamic mapping of a class of independent tasks onto heterogeneous computing systems, Journal of Distributed Computing, Special Issue on software support for distributed computing 59 (2).
[42] H. Izakian, A. Abraham, V. Snasel, Comparison of heuristics for scheduling independent tasks on heterogeneous distributed environments, in: Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on, Vol. 1, IEEE, 2009, pp. 8–12.
[43] M. Frigge, D. C. Hoaglin, B. Iglewicz, Some implementations of the boxplot, The American Statistician 43 (1) (1989) 50–54.
[44] G. Kamalam, V. M. Bhaskaran, A new heuristic approach:min-mean algorithm for scheduling meta-tasks on heterogenous computing systems, IJCSNS International Journal of Computer Science and Network Security 10.
[45] G. Kamalam, V. M. Bhaskaran, An improved min-mean heuristic scheduling algorithm for mapping independent tasks on heterogenous computing environment, INTERNATIONAL JOURNAL OF COMPUTATIONAL COGNITION (HTTP://WWW. IJCC. US) 8 (4) (2010) 85.
[46] D. D. H. Miriam, K. Easwarakumar, A double min min algorithm for task metascheduler on hypercubic p2p grid systems, International Journal of Computer Science Issues 7 (4) (2010) 8–18.
[47] K. Etminani, M. Naghibzadeh, A min-min max-min selective algorihtm for grid task scheduling, in: Internet, 2007. ICI 2007. 3rd IEEE/IFIP International Conference in Central Asia on, IEEE, 2007, pp. 1–7.
[48] X. He, X. Sun, G. Von Laszewski, Qos guided min-min heuristic for grid task scheduling, Journal of Computer Science

and Technology 18 (4) (2003) 442–451.

Table 6: Comparison of the results from the tried out approaches

Heuristic	Criteria value															
	Twt				Nwt				Ft				Cmax			
min-min	16,71				7,143				157,2				38,31			
max-min	22,06				8,138				195,8				38,83			
min-max	17,49				7,793				167,3				38,06			
sufferage	16,65				7,194				160,9				37,92			
search-based	9.533				5.34				140,8				36,79			
GP without opt.	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev
Twt opt.	**13,078**	**14,313**	**18,168**	**1,0301**	6,3978	6,9024	7,3991	0,2610	167,24	205,46	352,24	45,082	38,506	39,952	42,797	0,9138
Nwt opt.	14,126	23,996	54,672	9,6233	**6,3719**	**7,0464**	**7,6146**	**0,3077**	163,65	203,60	400,57	45,351	38,929	41,157	45,387	1,6619
Ft opt.	16,678	17,505	19,056	0,5857	6,9043	7,1657	7,4921	0,1375	**153,47**	**155,88**	**158,50**	**1,2409**	38,277	38,787	39,526	0,2601
Cmax opt.	16,942	20,750	36,740	3,8061	7,2890	7,8390	9,0244	0,3923	158,89	179,74	235,74	15,701	**37,924**	**38,358**	**38,908**	**0,2386**
GP	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev
Twt opt.	**12,962**	**13,657**	**14,624**	**0,4257**	6,1983	6,7119	7,2325	0,1963	170,54	204,25	353,24	42,755	38,689	37,774	42,237	0,7589
Nwt opt.	13,569	19,103	50,453	6,3062	**6,3842**	**7,0054**	**7,9387**	**0,3261**	158,74	192,56	337,00	39,731	38,536	40,306	43,398	1,3457
Ft opt.	16,406	17,116	18,670	0,4187	6,7633	7,1179	7,3686	0,1461	**153,97**	**155,29**	**158,56**	**0,8794**	38,315	38,688	39,158	0,1903
Cmax opt.	16,570	19,880	26,925	2,4381	7,2275	7,6990	8,2391	0,2844	158,41	175,72	203,07	12,016	**38,018**	**38,290**	**38,679**	**0,1504**
Dim. aw. GP	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev
Twt opt.	**13,132**	**13,778**	**14,808**	**0,4140**	6,4360	6,7674	7,3812	0,0204	173,50	188,70	385,41	30,679	38,749	39,513	43,153	0,6443
Nwt opt.	13,365	15,786	34,720	3,2765	**6,4899**	**6,9257**	**7,7344**	**0,2612**	170,99	192,35	348,92	25,469	38,727	40,338	44,994	1,1978
Ft opt.	16,323	17,256	18,653	0,5556	6,9541	7,1816	7,4758	0,1088	**153,39**	**155,89**	**160,57**	**1,7246**	38,139	38,629	39,179	0,2311
Cmax opt.	16,443	19,491	23,827	1,6382	7,2130	7,6812	8,4339	0,2528	156,50	172,91	192,26	7,9746	**38,033**	**38,268**	**38,929**	**0,1516**
GEP	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev
Twt opt.	**13,058**	**13,856**	**15,139**	**0,5552**	6,3117	6,7619	7,3343	0,2211	162,38	192,39	342,06	30,490	38,563	39,499	41,533	0,5221
Nwt opt.	13,675	17,617	32,895	4,0005	**6,4399**	**6,9413**	**7,5526**	**0,2486**	157,80	177,05	242,70	18,782	38,497	39,750	43,914	1,2769
Ft opt.	16,441	17,199	18,863	0,4958	6,9335	7,2047	7,4877	0,1418	**153,53**	**154,96**	**158,09**	**1,0420**	38,322	38,622	39,131	0,1634
Cmax opt.	16,740	20,270	27,330	2,5496	7,2318	7,6890	8,3596	0,2741	157,53	176,94	215,27	13,253	**37,945**	**38,217**	**38,729**	**0,1416**
GP with IDR	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev	min	avg	max	stdev
Twt opt.	**12,230**	**13,054**	**13,871**	**0,4025**	6,1760	6,6270	7,5124	0,2369	168,45	202,31	336,33	39,861	38,717	39,724	42,887	0,9236
Nwt opt.	13,272	20,887	71,042	10,910	**6,1519**	**6,7026**	**7,3131**	**0,2533**	159,29	195,32	313,60	36,833	38,366	40,211	44,041	1,5321
Ft opt.	16,360	17,274	19,232	0,5911	6,9268	7,1632	7,4445	0,1261	**151,56**	**154,07**	**156,86**	**1,0243**	38,218	38,595	39,171	0,2369
Cmax opt.	16,820	21,325	30,169	2,8461	7,0655	7,7872	8,4273	0,3183	155,35	182,60	219,52	12,330	**37,736**	**38,034**	**38,448**	**0,1821**