Towards Interpretable Dispatching Rules: Application of Expression Simplification Methods

Lucija Planinić
Faculty of Electrical Engineering and Computing
University of Zagreb
Email: lucija.planinic@fer.hr

Marko Đurasević
Faculty of Electrical Engineering and Computing
University of Zagreb
Email: marko.durasevic@fer.hr

Domagoj Jakobović
Faculty of Electrical Engineering and Computing
University of Zagreb
Email: domagoj.jakobovic@fer.hr

***Abstract*—Genetic programming (GP) is a powerful hyper-heuristic method used for evolving dispatching rules (DRs). DRs are commonly used to solve scheduling problems in which scheduling decisions have to be performed in a small amount of time, and are often based on incomplete information. Although GP is the most commonly used method for evolving DRs, it suffers from a serious problem called bloat, which represents the uncontrolled growth of expression trees during evolution. Bloat usually has two important repercussions on the evolved DRs. First, DRs become hard to understand and it is unclear by which strategy they perform scheduling decisions. Secondly, some trees can also include parts that are a result of overfitting on the training set and which reduce their generalization ability. To deal with the problem of bloating DRs, we propose a simplification method consisted of two parts: algebraic reduction and pruning. The simplification method is applied after the normal evolution process with GP is done to reduce the complexity of the evolved DRs. The results demonstrate that it is possible to reduce the number of nodes in an expression tree without significantly deteriorating its performance. This shows that the DRs evolved by GP are bloated and that substantial parts of them are redundant.**

I. Introduction

Automatically designing dispatching rules (DRs) for scheduling problems has become a viral research direction in the last several years [1], [2], [3]. Although a wide variety of methods can be used for generating new DRs, genetic programming (GP) received the most attention in this regard. Its popularity over other methods like artificial neural networks does not come only from a better performance, but also from their ability to provide symbolic expressions that can be interpreted [4]. However, GP suffers from a serious problem that is denoted as *bloat*, which is the uncontrolled growth of expression trees during the evolution. This has a negative effect on the interpretability of the evolved expressions, and as such it is not easy to determine the strategy by which the evolved rules perform different scheduling decisions.

Many methods have been used to limit the effects of bloat, some of which include limiting the maximum depth of the expression tree or using parsimony pressure [5]. However, it is difficult to specify restrictions on the size as it might not be known which expression complexity is required for solving a problem. Additionally, even if the size of the expression is constrained, this still does not provide a guarantee that the entire expression will be meaningful. Therefore, an alternative approach to increase the interpretability of DRs is to use methods which simplify the expression.

The goal of this paper is to apply two simplification methods to reduce the complexity of DRs evolved by GP. The first procedure performs an exact simplification, which means that the expression retains the same behaviour, i.e. for the same inputs the expression will calculate the same result before and after the simplification is performed. The second procedure allows a change in the behaviour of the DR by eliminating subexpressions which contribute the least to the solution quality. Both methods are applied to reduce the complexity of expressions evolved by GP. This means that GP is executed normally, without any changes, and after it evolves a DR, the two simplification procedures are applied to reduce its complexity. In that way the complexity of GP does not increase, and the method can be applied on already existing DRs to reduce their complexity.

The rest of the paper is organised as follows. Section II provides an overview of related works. Section III provides background on the considered problem, while section IV describes the automatic design of dispatching rules. In section V the proposed simplification procedure is described. Section VI describes the experimental setup, and in section VII the obtained results are displayed and discussed. Finally, in section VIII our conclusions are listed along with ideas for future work.

II. Related work

GP has been used to generate DRs for different scheduling problems, like the one machine environment [6], jobshop [7], unrelated machines [8], and resource constrained scheduling problem [9]. Recent years saw the emergence of several research directions in the area of automatically designing DRs, which include multi-objective optimisation [10], [11], ensemble learning methods [12], [13], [14], feature selection [15], [16], and many others. Although interpretability of DRs has been marked as an important research direction in [1], this line of research has still received very little attention.

In [17] the authors examine how the size of the evolved DRs correlates with its performance. The results demonstrate

This work has been supported in part by Croatian Science Foundation under the project "Hyperheuristic Design of Dispatching Rules", IP-2019-04-4333.

that as the size of the expression increases, the performance of the generated rules improves. However, as the sizes become larger, the improvements become smaller. This shows that the increase in the complexity of rules does not necessarily lead to a significant increase in the quality of DRs. In [18] the authors applied Dimensionally Aware GP (DAGP) [19] to evolve expressions which adhere to certain semantic rules. With DAGP it was possible to obtain dimensionally correct rules. However, this did not increase their intepretability. A similar direction to increase the interpretability of evolved rules was investigated in [20], where the authors applied a strongly typed grammar-based GP to constraint the search space to rules with a better interpretability. The results demonstrate that using the proposed method it was possible to obtain smaller DRs which were more interpretable than the ones obtained by standard GP. In [21] the authors use DAGP and propose a new interpretability measure which is added to the fitness for balancing between the quality and dimensional consistency of the evolved rules. The results demonstrate that the proposed method can achieve a trade off between effectiveness and interpretability of the evolved rules.

Until now, most studies in improving the interpretability of DRs were aimed at adapting the evolution process, which was mostly performed by including semantic rules. However, another approach to simplify DRs would be to remove redundant parts of the expression after it has been evolved. Such a process was performed manually in some studies to obtain rules which can be more easily examined [22], [23]. Naturally, performing simplification manually is a difficult and time consuming process. However, many procedures have been designed to automatically reduce the size and complexity of expressions.

The problem of simplifying expression trees has already been covered in many published studies. These studies have focused both on methods that simplify the expression tree without changing its performance compared to the original tree, and those that do change its performance. Methods for simplifying the expression tree without changing the original trees behaviour are usually based on performing algebraic simplification of the mathematical expression which the tree represents. Algebraic simplification is performed by applying algebraic rules to replace an expression by an equal expression which is simpler than the original. In [24] and [25] algebraic simplification was applied during evolution, rather than applying manual simplification after evolution. In [26] a new simplification method for symbolic regression was proposed. The proposed method replaces subtrees with known, simple subtrees if their evaluations over a set of regression points are the same.

The simplification methods that change the performance of the tree are based on the idea that not all parts of the tree contribute equally to the overall performance of the tree. This means that by removing the expressions that are not meaningful, the expression tree might become much simpler while the behaviour of the tree would not change much. These simplification methods differ by the techniques used to select which parts of the expression should be eliminated. In [27] a bloat control mechanism is proposed. The mechanism is based on evaluating the contribution of each function node to the expression tree and removing nodes without contribution before applying crossover to generate new individuals. In [28] a pruning scheme, called constant subtree pruning, is described. There, a subtree is replaced by its expected value calculated over a dataset if that replacement does not produce a statistically significant change in the output of the tree. A method for generating classification rules for the financial stock market is described in [29]. In the proposed method, rules are extracted from a decision tree, evaluated, and then pruned if they did not exceed the desired pruning threshold.

III. Problem definition

This study examines the parallel unrelated machines scheduling problem. In it, there are n jobs that need to be executed on one of the m available machines. The subscript j is used to denote a certain job, whereas the subscript i denotes a certain machine. For each job, several properties are defined. The processing time of job p_{ij} specifies the time that is required for machine i to process job j. In the considered environment it is presumed that there is no relation between the execution times of different jobs, or between jobs on different machines, and therefore the processing time for each job-machine pair needs to be specified. Each job has a release time r_j which defines the earliest moment at which a job can be considered for scheduling. A due date d_j, which specifies the moment until when a job needs to finish, is also defined for each job j. The due date is a soft constraint, which means that a job can finish after its due date, however, in that case it incurs a certain cost. Finally, all jobs do not have an equal importance, and are therefore associated with different weights w_j. Additional assumptions in the considered problem are that the jobs are not preemptive (the execution of a job cannot be interrupted), machines are always available, machines can execute one job at a time, and all jobs are independent and can be executed on any of the available machines.

Although many different scheduling criteria have been considered for the unrelated parallel machines environment, in this paper we consider the minimisation of the total weighted tardiness criterion (TWT). The TWT is defined as in equation (1), where T_j is defined as in equation (2).

$$TWT = \sum w_j T_j \tag{1}$$

$$T_j = \max(C_j - d_j, 0) \tag{2}$$

IV. Automatic design of DRs

DRs are simple heuristics which build schedules by deciding which job should be scheduled on which machine each time at least one machine is empty and there is at least one job that needs to be scheduled. Dispatching rules consist out of two parts - a priority function and a schedule generation scheme.

The schedule generation scheme chooses which job-machine pair is going to be selected next for scheduling based on the priorities calculated for each available job-machine

pair. The pseudocode for the schedule generation scheme is denoted in Algorithm 1. Since the schedule generation scheme is generally quite simple, it is usually defined manually.

Algorithm 1: Schedule generation scheme used by generated DRs

```
while unscheduled jobs are available do
    Wait until at least one job and machine are available
    for all available jobs j and each machine i in m do
        Get the priority π_ij of scheduling j on machine i
    end for
    for all available jobs do
        Determine the machine with the best π_ij value
    end for
    while jobs whose best machine is available exist do
        Determine the best priority of all such jobs
        Schedule the job with best priority
    end while
end while
```

However, the method by which the priority of a job-machine pair is calculated is not defined within the schedule generation scheme. The expression used to calculate priorities of job machine pairs is called a priority function. Since it is challenging to manually define a priority function for a specific problem, this process is usually automated with the use of genetic programming. Genetic programming evolves programmes or mathematical expressions by using individuals represented by expression trees. Expression trees consist out of functions and terminals. Terminal nodes are the leaf nodes in an expression tree. They provide specific information about the scheduling problem which is being solved. The terminals that were used are listed in Table I. The function set consisted out of 5 different functions: addition, subtraction, multiplication, protected division which returns one in case of division by zero, and the unary positive operator which returns zero if the provided value is negative and returns the provided value otherwise. Both sets were selected based on previous experiments [18].

V. Simplification procedures

In this study, two simplification procedures were used to reduce the complexity of the evolved DRs: the reduction procedure which does not change the behaviour of the expression, and pruning which allows a change in the behaviour of the DR. These simplification procedures were applied to the best trees obtained by different GP runs.

TABLE I: Terminal set

Terminal	Description
pt	processing time of job j on machine i (p_{ij})
$pmin$	minimal processing time (MPT) of job j
$pavg$	average processing time of job j on all the machines
PAT	time until machine with the MPT for job j becomes available
MR	time until machine i becomes available
age	time which job j spent in the system
dd	time until which job j has to finish with its execution (d_j)
w	weight of job j (w_j)
SL	slack of job j, $-max(d_j - p_{ij} - t, 0)$

A. Reduction

The goal of the reduction simplification procedure is to change the expression in a way that its behaviour remains completely the same. This is done by traversing the tree and identifying patterns of expression which can be replaced by a smaller expression with the same behaviour. For example, the expression $x + 0$ can be replaced by x without any loss in meaning. The method which is applied for reduction is based on the Compare-Match algorithm which is proposed in [30]. In this algorithm the entire expression tree is traversed and parts of the expression are matched against a set of predefined reduction rules. The set of the reduction rules that are applied for simplification is given by equations 3 to 23. These rules include simple mathematical rules which eliminate neutral elements for certain operations, or apply simple mathematical rules to simplify the expression. It needs to be stressed that the variables X, Y, and Z do not only represent an input variable, but can represent an expression (subtree) of any kind. This allows the reduction method to match the rules to any part of the expression. Since it is possible that after applying one reduction rule it would be possible to apply other rules, after the entire tree is traversed the entire procedure is repeated if at least one simplification rule was applied. Therefore, the procedure terminates when in a single pass through the tree no simplification was performed.

$$+ * XY * XZ \rightarrow *X + YZ \tag{3}$$
$$+ * YX * ZX \rightarrow *X + YZ \tag{4}$$
$$*X1 \rightarrow X \tag{5}$$
$$*1X \rightarrow X \tag{6}$$
$$*X0 \rightarrow 0 \tag{7}$$
$$*0X \rightarrow 0 \tag{8}$$
$$+0X \rightarrow X \tag{9}$$
$$+X0 \rightarrow X \tag{10}$$
$$/XX \rightarrow 1 \tag{11}$$
$$-XX \rightarrow 0 \tag{12}$$
$$-X0 \rightarrow X \tag{13}$$
$$/X1 \rightarrow X \tag{14}$$
$$+X * YX \rightarrow * + 1YX \tag{15}$$
$$+X * XY \rightarrow * + 1YX \tag{16}$$
$$+ * YXX \rightarrow * + 1YX \tag{17}$$
$$+ * XYX \rightarrow * + 1YX \tag{18}$$
$$/ * XYY \rightarrow X \tag{19}$$
$$*/XYY \rightarrow X \tag{20}$$
$$-X - 0Y \rightarrow +XY \tag{21}$$
$$+X - -1Y1 \rightarrow -XY \tag{22}$$
$$-X - -1Y1 \rightarrow +XY \tag{23}$$

B. Pruning

The goal of pruning is to reduce the size of the expression by removing parts of the expression which are deemed to contribute the least to the quality of the expression. This is performed in a way that the entire tree is traversed and each node is replaced by a *neutral element*. A neutral element is defined as an element which does not have an influence on the calculation of an operator. For summation and subtraction 0 represents a neutral element, whereas for multiplication and division the neutral element is 1. For the positive operator there is no neutral element, since it has only one operand. By replacing a subtree with a neutral element, it is effectively excluded from the expression, whereas the rest of the expression is kept equal to the original.

Removing parts of the expression in such a way can have a significant impact on the semantic of the original expression. Therefore, the expression obtained after the simplification needs to be evaluated. This is done by evaluating the expression on a given set of problem instances and comparing the fitness of the original expression with the simplified one. Although different strategies can be used to accept or reject the simplified expression, here we use a simple method which is based on a relative fitness change (RFC). This measure is calculated as in equation (24), where f_o represents the fitness of the original expression, whereas f_s represents the fitness of the simplified expression. The simplification is accepted only if RFC is smaller than a given pruning threshold specified by the user. Similar as with reduction, this method is also applied iteratively. Each time a part of the expression is eliminated, the procedure starts from the root node, and traverses the entire tree once again. This process is repeated until no further simplification is accepted during a single traversal of the expression.

$$RFC = \frac{f_s - f_o}{f_o} \tag{24}$$

C. Complete simplification procedure

The simplification procedure used in this paper represents a combination of the previous two methods, which are then invoked on a given solution. The simplification procedure consists out of three steps. In the first step the reduction method is applied to simplify the initial solution algebraically. After that, pruning is applied to detect unnecessary parts in the expression obtained as the result of the first step. Finally, another round of reduction is performed to eliminate any terminal nodes that were introduced in the previous step.

Since the reduction method is less computationally expensive, it is applied first to reduce the size of the expression and to reduce the number of nodes that need to be pruned in the pruning phase. Since pruning introduces neutral elements in the expression, the reduction method is applied once again to remove them from the expression. The obtained result represents the smallest expression which can be acquired by these methods for the given pruning threshold. Any further applications of the simplification procedures would terminate immediately since no additional rules can be used for further reductions.

VI. Experimental setup

To evolve the DRs, a standard steady state tournament GP variant with 3-tournament selection was used. The algorithm used a population size of 1000 individuals, a mutation probability of 0.3, and 80000 function evaluations as the stopping condition. All parameter values were previously optimised [18]. To investigate the effect of the simplification procedure on DRs of different sizes, GP was executed with tree depths of 3, 5, 7, 9, 11, 13. For each of the considered tree depths, 30 executions were performed to obtain statistically significant results. The optimisation criterion was the total weighted tardiness (TWT) criterion. The training and evaluation of the expressions were performed on two independent problem instance sets, both consisting out of 60 instances.

After the 30 DRs have been evolved by GP, the simplification procedure was applied on each of the evolved rules. To examine the performance of the simplification method, several pruning thresholds have been used: 0.01, 0.05, 0.1, 0.15, and 0.2. The simplification procedure was performed using the test set to evaluate the simplified expression to determine whether it should be accepted or not. It should be outlined that the results for the tree depth of 13 and pruning threshold 0.01 could not be obtained due to a long execution time for several trees which consisted of around thousand nodes. To examine whether a statistically significant difference exists between the original and simplified expressions, the Mann-Whitney statistical test was performed. The results were considered to be significant if the obtained p-value was lower than 0.05.

VII. Results and discussion

Table II represents the results obtained on the test set. The bold values denote that, for the tested pruning threshold values, the simplified DRs achieved significantly better or equally good results as the DRs prior to simplification. From that, several interesting phenomena can be observed. Generally, for the two lowest pruning threshold values the fitness of the rules stays the same or even improves after the simplification. This indicates that the evolved expressions started bloating and include unnecessary parts in parts which can be removed without any consequences, or that the trees even started to overfit on the training set. Therefore, slightly pruning such DRs is shown to be extremely beneficial.

When using pruning thresholds of 0.1 and 0.15 the simplified expressions start to perform worse than the original expressions for tree depths 5 and 7. For other tree depths the results are still at least equally good as those obtained by the original expressions. An interesting observation can be made for the expressions generated with depth 3, in which case the results significantly improved when using larger threshold values. It seems that in the case of small tree depths the expressions overfit too easily on the training set. However, the generated rules contain valuable knowledge, which becomes clear when the expressions are further simplified. Larger tree

TABLE II: Test set fitness results.

Pruning thresh			**0.01**	**0.05**	**0.1**	**0.15**	**0.2**
depth		**original**	**final**	**final**	**final**	**final**	**final**
3	**min**	13.25	13.25	13.30	13.30	13.30	13.38
	med	14.21	**14.21**	**14.21**	**13.44**	**13.44**	**13.48**
	max	14.91	14.91	14.91	16.07	16.07	16.07
5	**min**	12.88	13.01	13.01	13.24	13.24	13.44
	med	13.94	**13.93**	**13.75**	14.23	14.29	14.64
	max	14.76	14.95	14.91	15.18	16.14	18.61
7	**min**	13.05	13.06	12.68	13.00	12.86	13.38
	med	13.98	**14.00**	**14.01**	**14.23**	14.50	14.54
	max	17.35	20.99	24.49	23.67	15.86	19.28
9	**min**	12.80	12.87	12.92	13.05	13.46	13.46
	med	14.24	**14.15**	**14.18**	**14.23**	**14.44**	14.82
	max	18.20	17.74	16.46	16.04	16.02	16.68
11	**min**	12.92	12.93	12.89	12.90	13.31	13.43
	med	14.28	**14.25**	**14.22**	**14.06**	**14.22**	**14.47**
	max	328.1	46.05	46.05	46.05	46.05	46.05
13	**min**	12.58	-	12.99	13.09	13.30	13.30
	med	14.40	-	**14.27**	**14.30**	**14.33**	**14.68**
	max	18.34	-	21.14	20.85	19.73	20.84

depths also benefit from a larger pruning threshold, which also seems to indicate that the evolved expressions contain parts that are simply a consequence of overfitting on the training set. Since for tree depths of 5 and 7 larger pruning threshold values lead to deterioration of the results, this might indicate that these tree depths are appropriate for this problem and thus be less susceptible to including unnecessary parts.

For the largest pruning threshold value it can be seen that the results deteriorate for most tree depths, although in some cases they are still statistically equally good to those obtained by the original DRs. However, these results indicate that at this point the procedures started to remove important genetic material, which then lead this expressions to under perform.

Aside from the results on the test set, it is also interesting to observe the results on the training set that are presented in Table III. These results show how the fitness deteriorates more and more as the pruning threshold is increased. This means that the methods remove parts of the DR which directly influence its performance on the test set. However, as on the test set the deterioration is not as prominent, we can conclude that the method was efficient in removing parts of the expression which did not contain general knowledge, but rather parts which overly specialised for the test set.

Table IV denotes the expression sizes (in the number of nodes) that were obtained by GP and after each simplification procedure. The column denoted as "original" denotes the size of the original expression. The first "reduce" column represents the expression size obtained after the initial reduction. Since this procedure has no parameters, it always produces the same result. The remaining columns show the results obtained by pruning for different pruning threshold values. After pruning the reduction method is again performed to eliminate any neutral elements that were introduced in the expression during pruning.

The table shows that the very first application of reduction had very little effect on the size of the expressions. This means that the original expression had very little parts that could be eliminated without affecting the semantic of the expression. However, pruning shows a much better ability to reduce the size of the expression. Naturally, the pruning threshold has a large effect on how much the expressions can be reduced. However, even for the smallest pruning threshold value the expressions can be reduced by around 30% in some cases. As the pruning threshold increases the efficiency of these methods also increases, especially for the larger tree depths where the expressions were regularly reduced to expressions containing between 10 and 20 nodes. Naturally, this does come with a negative effect on the fitness of the evolved expressions. The table also shows that the additional reduction step after pruning is important as it does significantly reduce the size of the expression. This shows that pruning removed a lot of sub expressions and introduced a large amount of neutral elements.

Figure 1 shows an example of how an evolved PF is simplified with reduction and pruning. Subfigure 1a shows the original expression which consists of 25 nodes. The first step which is performed is applying the reduction method, which in this case cannot reduce the expression as it cannot match any of the reduction rules to any of the subexpressions in the tree. In the next step, pruning is performed and the procedure found an expression which it then replaces with 0 (since it parent node represents the summation operator). The result of this pruning step is denoted in subfigure 1b. After this subexpression has been pruned, the pruning steps restarts from the root and finds that it can remove another

TABLE III: Train set fitness results.

Pruning thresh			**0.01**	**0.05**	**0.1**	**0.15**	**0.2**
depth		**original**	**final**	**final**	**final**	**final**	**final**
5	**min**	14.05	14.05	14.28	15.04	15.21	15.47
	med	14.66	14.76	15.26	15.88	16.13	16.51
	max	15.20	15.28	15.91	16.49	17.17	17.88
9	**min**	13.88	14.00	14.24	15.00	15.24	15.24
	med	14.40	14.52	15.03	15.58	16.21	16.57
	max	15.07	15.11	15.78	16.35	17.26	17.69
11	**min**	13.61	13.73	14.22	14.77	15.30	15.30
	med	14.41	14.54	15.00	15.62	16.15	16.58
	max	353.8	46.13	46.13	46.13	46.13	46.13
13	**min**	13.69	-	14.35	15.04	15.33	15.54
	med	14.44	-	15.12	15.69	16.11	16.47
	max	15.65	-	16.36	17.11	17.99	18.58
7	**min**	14.02	14.16	14.59	15.11	15.11	15.49
	med	14.47	14.53	15.10	15.62	16.16	16.52
	max	14.88	23.72	26.42	29.56	16.94	17.77
3	**min**	15.07	15.07	15.07	15.07	15.36	15.36
	med	15.44	15.44	15.44	16.55	16.55	16.55
	max	15.78	15.78	16.01	16.76	16.88	18.44

part of the expression, and the result is shown in subfigure 1c. In the next pass of the reduction procedure, no part can be further removed without significantly reducing the quality of the expression, and therefore the pruning procedure is terminated. Now the reduction procedure is applied once again to remove any neutral elements and unneeded operators from the expression. The final result is shown in subfigure 1d, which shows that the 0 constant and its parent operator were removed from the expression. The final expression has only 9 nodes. In comparison to the original size, we can see that the simplification procedures reduces the expression to almost a third of its original size.

An important property of the simplification methods is their execution time. This is especially true for pruning since it has to perform a lot of function evaluations during its execution. Table V outlines the average execution time in seconds required to execute pruning with different threshold values for a single DR. For smaller tree sizes, the method can be executed in less than a minute. However, as the sizes increase, it is evident that the execution times become too large as there are too many combinations that need to be tested. This naturally represents a limit of the applied procedure. However, in the context of DRs, it is highly unlikely that DRs of such large sizes would even be evolved. A similar situation is evident for the threshold values, since for larger values the execution time decreases as it allows the procedure to eliminate larger portions of expressions. The lowest threshold value required the most time. However, since the previous results demonstrated that it does not offer any benefits in comparison to some larger values, they can be used to decrease the execution time without any repercussions on the performance of DRs.

VIII. Conclusion

This paper considers the problem of simplifying automatically generated dispatching rules. The results have demonstrated that when using the considered simplification procedures, it is possible to reduce the number of nodes in the expression, and as a consequence, also improve their interpretability. Out of the applied simplification procedures, pruning was shown to be more appropriate as the evolved rules usually did not include parts which could be algebraically reduced. The results also show that significantly reducing the size of dispatching rules does not necessarily result in significantly worse solution. This demonstrates that the evolved expressions are bloated, and that significant parts of them are redundant and therefore do not have any effect on the scheduling decisions that are performed.

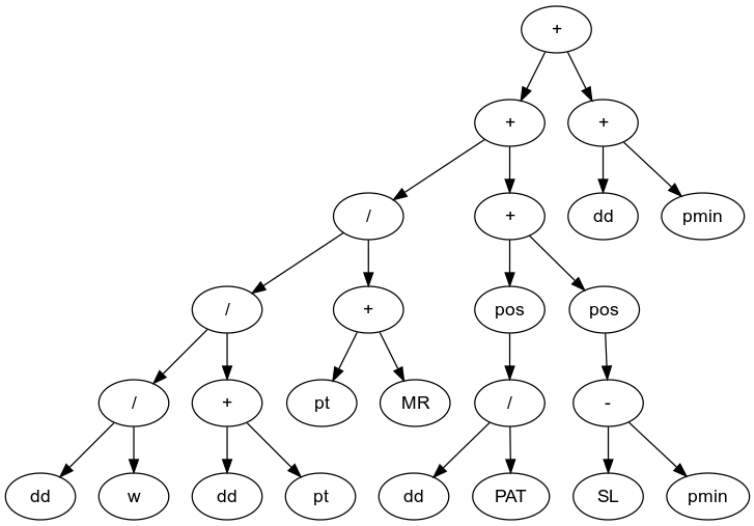
Future work could be focused on employing the proposed simplification procedures online, during the evolution process. The goal in this line of research would be to determine the best strategy by which they should be employed (for example how often and on which individuals to apply simplification during evolution). In addition, as the pruning methods execution time is high, especially for smaller parameter values, another research direction would be to examine possibilities of reducing the number of tested combinations in this simplification step. Finally, the discussed simplification methods were applied to expressions generated to optimise the TWT criterion. The simplification methods could be applied to expressions generated to optimise other objectives or to expressions designed for multi-objective problems.

References

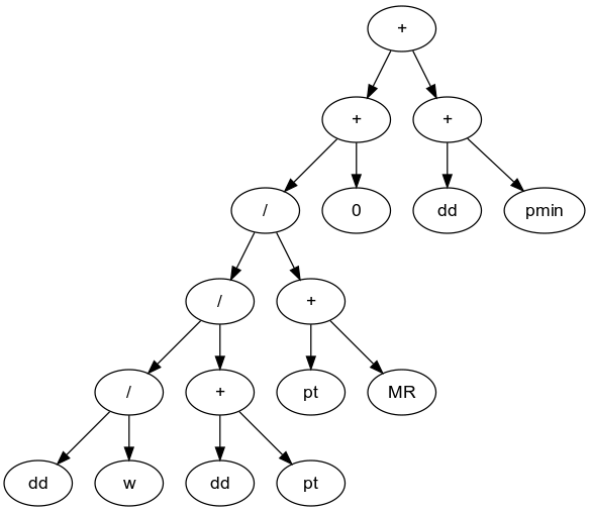
[1] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated Design of Production Scheduling Heuristics: A Review," *IEEE Transactions on*

TABLE IV: Size results.

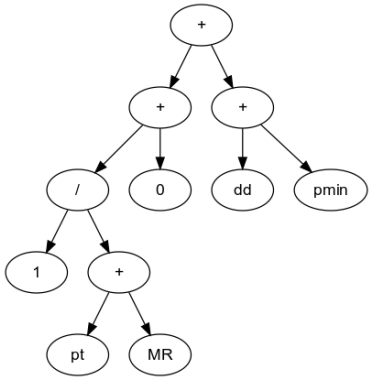
Pruning thresh				0.01		0.05		0.1		0.15		0.2	
Depth		**original**	**reduce**	**prune**	**reduce**	**prune**	**reduce**	**prune**	**reduce**	**prune**	**reduce**	**prune**	**reduce**
3	**min**	12	11	11	11	11	10	11	9	11	9	11	9
	med	13	13	13	13	13	13	12	11	12	11	11.5	11
	max	15	15	15	15	15	15	15	15	15	15	15	15
5	**min**	25	25	20	18	16	13	11	9	11	9	11	9
	med	42	41.5	35	34	28.5	24	22.5	19	20.5	16	19	13.5
	max	59	57	50	50	43	41	34	29	34	26	35	27
7	**min**	31	31	27	25	23	21	17	12	16	11	15	10
	med	88	85.5	64	60.5	44.5	36.5	31.5	24.5	27	19	25	17
	max	155	145	107	99	68	64	59	49	43	31	47	37
9	**min**	42	40	34	30	19	17	17	13	14	9	14	9
	med	128	118.5	94	86	58	49.5	37	28	31	20	28	18
	max	354	328	207	193	135	113	96	74	67	47	54	39
11	**min**	53	53	3	3	3	3	3	3	3	3	3	3
	med	167.5	161.5	103.5	92.5	62.5	50.5	34	26.5	23.5	17	20	14.5
	max	602	571	287	249	181	169	103	89	64	54	73	58
13	**min**	79	79	-	-	17	15	15	11	12	11	12	9
	med	304	298.5	-	-	76.5	61.5	34	27	24.5	19	22.5	15
	max	1909	1862	-	-	352	260	269	205	244	180	175	113



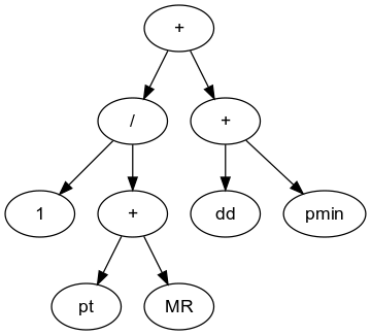
(a) Original tree



(b) Tree after first pruning step



(c) Tree after second pruning step



(d) Final tree after reduction

Fig. 1: The process of simplification of an individual

TABLE V: Average simplification time in seconds

Depth		3	5	7	9	11	13
Pruning thresh	0.01	0.926	3.428	48.430	112.709	221.220	-
	0.05	0.606	4.840	22.789	57.939	116.417	3527.585
	0.1	1.194	4.504	20.898	42.983	68.477	807.172
	0.15	0.715	4.285	16.491	34.039	51.535	419.994
	0.2	0.687	4.161	13.743	36.593	50.885	414.218

Evolutionary Computation, vol. 20, no. 1, pp. 110–124, Feb. 2016.

[2] S. Nguyen, Y. Mei, and M. Zhang, "Genetic programming for production scheduling: a survey with a unified framework," *Complex & Intelligent Systems*, vol. 3, no. 1, pp. 41–66, Mar. 2017.

[3] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, *Genetic Programming for Job Shop Scheduling*. Cham: Springer International Publishing, 2019, pp. 143–167.

[4] J. Branke, T. Hildebrandt, and B. Scholz-Reiter, "Hyper-heuristic Evolution of Dispatching Rules: A Comparison of Rule Representations," *Evolutionary Computation*, vol. 23, no. 2, pp. 249–277, Jun. 2015.

[5] R. Poli and N. F. McPhee, "Parsimony pressure made easy: Solving the problem of bloat in GP," in *Theory and Principled Methods for the Design of Metaheuristics*. Springer Berlin Heidelberg, Nov. 2013, pp. 181–204. [Online]. Available: https://doi.org/10.1007/978-3-642-33206-7_9

[6] F. J. Gil-Gala, C. Mencía, M. R. Sierra, and R. Varela, "Evolving priority rules for on-line scheduling of jobs on a single machine with variable capacity over time," *Applied Soft Computing*, vol. 85, p. 105782, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494619305630

[7] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "A computational study of representations in genetic programming to evolve dispatching rules for the job shop scheduling problem," *IEEE Transactions on Evolutionary Computation*, vol. 17, no. 5, pp. 621–639, 2013.

[8] K. Jaklinović, M. Đurasević, and D. Jakobović, "Designing dispatching rules with genetic programming for the unrelated machines environment with constraints," *Expert Systems with Applications*, vol. 172, p. 114548, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417420311921

[9] M. Đumić, D. Šišejković, R. Čorić, and D. Jakobović, "Evolving priority rules for resource constrained project scheduling problem with genetic programming," *Future Generation Computer Systems*, vol. 86, pp. 211–221, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X1732441X

[10] M. Durasević and D. Jakobović, "Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment," *Genetic Programming and Evolvable Machines*, Sep 2017. [Online]. Available: https://doi.org/10.1007/s10710-017-9310-3

[11] F. Zhang, Y. Mei, and M. Zhang, "Evolving dispatching rules for multi-objective dynamic flexible job shop scheduling via genetic programming hyper-heuristics," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2019, pp. 1366–1373.

[12] M. Durasević and D. Jakobović, "Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment," *Genetic Programming and Evolvable Machines*, vol. 19, no. 1, pp. 53–92, Jun 2018. [Online]. Available: https://doi.org/10.1007/s10710-017-9302-3

[13] J. Park, Y. Mei, S. Nguyen, G. Chen, and M. Zhang, "An investigation of ensemble combination schemes for genetic programming based hyper-heuristic approaches to dynamic job shop scheduling," *Applied Soft Computing*, vol. 63, pp. 72 – 86, 2018.

[14] M. Djurasević and D. Jakobović, "Creating dispatching rules by simple ensemble combination," *Journal of Heuristics*, May 2019. [Online]. Available: https://doi.org/10.1007/s10732-019-09416-x

[15] Y. Mei, S. Nguyen, B. Xue, and M. Zhang, "An efficient feature selection algorithm for evolving job shop scheduling rules with genetic programming," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 5, pp. 339–353, 2017.

[16] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Genetic programming with adaptive search based on the frequency of features for dynamic flexible job shop scheduling," in *Evolutionary Computation in Combinatorial Optimization*, L. Paquete and C. Zarges, Eds. Cham: Springer International Publishing, 2020, pp. 214–230.

[17] E. Pitzer, A. Beham, M. Affenzeller, H. Heiss, and M. Vorderwinkler, "Production fine planning using a solution archive of priority rules," in *3rd IEEE International Symposium on Logistics and Industrial Informatics*, 2011, pp. 111–116.

[18] M. Durasevic, D. Jakobović, and K. Knežević, "Adaptive scheduling on unrelated machines with genetic programming," *Applied Soft Computing*, vol. 48, pp. 419–430, Nov. 2016.

[19] M. Keijzer and V. Babovic, "Dimensionally aware genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds., vol. 2. Orlando, Florida, USA: Morgan Kaufmann, 13-17 Jul. 1999, pp. 1069–1076. [Online]. Available: http://gpbib.cs.ucl.ac.uk/gecco1999/GP-420.ps

[20] R. Hunt, J. Richard, and M. Zhang, "Evolving Dispatching Rules with Greater Understandability for Dynamic Job Shop Scheduling Mark Johnston," no. x, 2016. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.702.1375

[21] Y. Mei, S. Nguyen, and M. Zhang, "Constrained dimensionally aware genetic programming for evolving interpretable dispatching rules in dynamic job shop scheduling," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10593 LNCS, pp. 435–447, 2017.

[22] M. Đurasević and D. Jakobović, "Evolving dispatching rules for optimising many-objective criteria in the unrelated machines environment," *Genetic Programming and Evolvable Machines*, vol. 19, no. 1-2, pp. 9–51, Sep. 2017. [Online]. Available: https://doi.org/10.1007/s10710-017-9310-3

[23] M. Đurasević and D. Jakobović, "Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment," *Applied Soft Computing*, vol. 96, p. 106637, 2020.

[24] P. Wong and M. Zhang, "Algebraic simplification of gp programs during evolution," in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 927–934. [Online]. Available: https://doi.org/10.1145/1143997.1144156

[25] M. Zhang, P. Wong, and D. Qian, "Online program simplification in genetic programming," in *Simulated Evolution and Learning*, T.-D. Wang, X. Li, S.-H. Chen, X. Wang, H. Abbass, H. Iba, G.-L. Chen, and X. Yao, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 592–600.

[26] N. Mori, R. McKay, N. Hoai, D. Essam, and S. Takeuchi, "A new method for simplifying algebraic expressions in genetic programming called equivalent decision simplification," in *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, vol. 13, 06 2009, pp. 237–244.

[27] A. Song, D. Chen, and M. Zhang, "Bloat control in genetic programming by evaluating contribution of nodes," in *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 1893–1894.

[28] P. Rockett, "Pruning of genetic programming trees using permutation tests," *Evolutionary Intelligence*, vol. 13, no. 4, pp. 649–661, Dec 2020. [Online]. Available: https://doi.org/10.1007/s12065-020-00379-8

[29] A. Garcia-Almanza and E. Tsang, "Simplifying decision trees learned by genetic programming," 01 2006, pp. 2142 – 2148.

[30] J.-M. Steyaert and P. Flajolet, "Patterns and pattern-matching in trees: An analysis," *Information and Control*, vol. 58, no. 1, pp. 19–58, 1983. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0019995883800564