On the Application of ϵ-Lexicase Selection in the Generation of Dispatching Rules

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

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

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

***Abstract*—Dynamic online scheduling is a difficult problem which commonly appears in the real world. This is because the decisions have to be performed in a small amount of time using only currently available incomplete information. In such cases dispatching rules (DRs) are the most commonly used methods. Since designing them manually is a difficult task, this process has been successfully automatised by using genetic programming (GP). The quality of the evolved rules depends on the problem instances that are used during the training process. Previous studies demonstrated that careful selection of problem instances on which the solutions should be evaluated during evolution improves the performance of the generated rules. This paper examines the application of the ϵ-lexicase selection to the design of DRs for the unrelated machines scheduling. This selection offers a better solution diversity since the individuals are selected based on a smaller subset of instances, which leads to the creation of DRs that perform well on the selected instances. The experiments demonstrate that this type of selection can significantly improve the results for the Roulette Wheel and Elimination GP variants, while achieving the same performance as the Steady State Tournament GP. Furthermore, the ϵ-lexicase based algorithms have a better convergence rate, which means that the increased diversity in the population has a positive effect on the evolution process.**

***Index Terms*—lexicase, genetic programming, dispatching rules, scheduling**

I. Introduction

Scheduling problems, in which a set of jobs has to be allocated to a scarce set of resources, appear in many everyday situations, such as workforce scheduling [1], manufacturing [2], nurse rostering [3], and similar. Various scheduling problems became widely studied by practitioners to provide algorithms which could solve them by optimising one or multiple criteria. However, most scheduling problems are NP-hard, which means that they cannot be solved to optimality in a reasonable amount of time [4]. Because of that, a wide range of heuristic and metaheuristic methods have been proposed and applied for solving scheduling problems [5].

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

978-1-7281-8393-0/21/$31.00 ©2021 IEEE

Heuristic methods that are used to solve scheduling problems are usually divided into two groups, improvement and constructive heuristics. Improvement heuristics start with one or more existing, usually randomly generated, solutions which they improve upon. Various metaheuristic algorithms, like genetic algorithms or differential evolution, belong into this group [6], [7]. However, these methods are suitable for static scheduling problems, i.e. problems in which all the information is known beforehand, since they search the entire solution space. In real world situations, scheduling problems are often dynamic in their nature, meaning that it is not always known when jobs which need to be scheduled will appear or what their characteristics will be [8]. As such, dynamic problems usually cannot be solved with improvement methods since not all information is available and it is not possible to search the entire solution space. In those cases, the most commonly used methods are constructive heuristics, which start with an empty schedule and iteratively construct the complete solution. These methods are most often defined in the form of dispatching rules (DRs), which are simple heuristics that use a priority function (PF) to rank the available jobs and schedule the one with the best priority [9], [10]. Because they only calculate the priority of available jobs, which can be calculated quickly, DRs represent the most commonly used methods for solving scheduling problems under dynamic conditions.

DRs, however, come with a certain drawback. Unlike metaheuristics which are general and can easily be adapted and applied to various scheduling problems, DRs are highly problem specific. This means that for each scheduling problem variant and optimisation criterion a different DR may need to be defined. This becomes evident when it is required to optimise non-standard criteria defined by users for which a suitable DR might not exist [11]. As a consequence, a lot of research is focused on the means of automatically generating DRs. Although methods like neural networks [12] were applied for generating DRs, genetic programming (GP), which is a hyper-heuristic method [13], became the most commonly used one for the automatic design of DRs [14]. Until now a lot of studies were done in the area of automatically designing new DRs [14]–[16], with recent studies focusing on evolving multi-objective problems [17], [18], applying ensemble methods [8],

[19], [20], feature selection [21], [22] and similar.

One problem associated with the automatic development of hyper-heuristics is the construction of the training set that will be used by GP. The choice of problem instances the training set will consist of has a significant influence on many aspects of the training process. First of all, meaningful instances have to be selected to provide enough information for the GP to be able to generate efficient DRs. The instances also need to be diverse enough to allow GP to generate DRs which can react to different situations. On the other hand, since each solution needs to be evaluated on the entire training set, the number and size of problem instances has to be restricted to ensure that the DRs can be generated in a reasonable amount of time. As a consequence, several studies focused on the design of the training set and its usage during the training process.

In [23] the authors examined how using a different number of problem instances during learning had an effect on the performance of the evolved DRs and demonstrated that in their settings it was best to use only a single problem instance per generation, but selecting a new instance in each generation. An interesting approach applied for the generation of DRs is the use of surrogate models to evaluate the solutions [24], [25]. Applying such models showed that it is possible to achieve either a better performance than with the regular evaluation, or a similar performance but in a smaller amount of time. In another study, an active sampling method was coupled with GP to select good training instances during evolution [26]. The paper demonstrated that this approach provides a better balance between exploitation and exploration. A similar approach was considered in [27] where the authors sampled instances which exhibited potential of improving the Pareto front when designing DRs for the multi-objective problems.

A somewhat less represented line of research considers the influence of evolutionary selection on the effectiveness of the evolved heuristics. Since hyper-heuristic optimization inevitably involves the application of an accompanying evolutionary algorithm, the choice of the method of selection will certainly affect the convergence properties, but can also influence the generalization abilities of the priority functions. Because the evaluation of potential solutions in GP usually relies on a set of problem instances, this was the motivation for the development of the lexicase selection [28], which differentiates potential solutions by their ability to successfully solve individual problem-specific test cases. In the symbolic regression domain, the method was extended to handle non-discrete results and is known as ϵ-lexicase [29]. This selection technique naturally lends itself to the problem of learning priority functions; instead of using a single cumulative measure over the whole training set, as is the most common approach, the effectiveness of priority rules may be estimated more efficiently by observing their performance on individual training instances.

The idea of the ϵ-lexicase selection is to use a subset of problem instances to select individuals. In that way, this selection provides a larger diversity in the population since different solutions can perform well when only several problem instances are considered. As a consequence, different individuals will have a chance to participate in the mating process and increase the diversity of the population. The benefit of the ϵ-lexicase selection is that it can be used by classical GP methods instead of other selection types (such as fitness proportionate, random or tournament selection).

The aim of this study is to investigate the influence of different selection mechanisms in the evolution of priority functions and to extend the algorithms with the use of ϵ-lexicase. For that purpose the ϵ-lexicase selection will be used in combination with three standard GP variants for the design of dispatching rules in the unrelated machines environment.

The rest of the paper is organised as follows. Section II provides the background on the considered problem and the applied methods. Section III describes the experimental setup and outlines the obtained results. A deeper analysis and discussion about the obtained results is given in Section IV. Finally, Section V outlines the conclusion and gives directions for future research.

II. Background and Methodology

A. Problem Definition

The unrelated machines environment consists out of n jobs which need to be scheduled on one of the available machines m [4]. A concrete job is denoted with the index j, while a concrete machine is denoted with the index i. Each machine can execute only a single job at a time and no preemption is allowed once a job starts executing. Each job is defined by the following properties:

- p_{ij} - processing time of job j on machine i,
- r_j - release time of job j,
- d_j - due date of job j,
- w_j - weight (importance) of job j.

The criterion which is optimised is the total weighted tardiness, which is defined as $TWT = \sum_j w_j T_j$, where T_j represents the tardiness of job j. Tardiness is defined as the amount of time that a job spent executing after its due date, i.e. $T_j = \max(C_j - d_j, 0)$, where C_j represents the completion time of job j.

The problem is considered under dynamic conditions. This means that information about the jobs becomes available only when they are released into the system. The moments in which jobs will appear in the system are also not known beforehand.

B. Automatic Design of Dispatching Rules

DRs usually consist of two parts, the priority function (PF) and the schedule generation scheme (SGS). The SGS defines when the DRs perform the selection of jobs and how they allocate them to the machines [30]. The SGS that is used by automatically designed DRs is denoted in algorithm 1. The SGS works in a way that it performs the scheduling decision each time a job and a machine are available. When this happens, the SGS calculates the priority value for scheduling each available job on each of the machines. The job-machine pair with the highest priority value is then selected and the job is scheduled on the corresponding machine.

Algorithm 1: SGS 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

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, $-max(d_j - p_{ij} - t, 0)$

As the SGS is usually quite simple, it is defined manually. However, the SGS does not specify by which strategy it will prioritise jobs and machines. For this it uses a PF to calculate the priorities of jobs and machines. Since it is difficult to manually define a PF, this process is delegated to GP which by using a set of problem instances constructs a PF that performs well over those instances. To allow GP to design DRs, the nodes that construct the expression trees need to be specified. The applied terminal nodes are listed in Table I. These nodes provide specific information about the considered problem like the processing times or due dates of jobs. In addition, function nodes that are used to construct the expression also have to be defined. Function nodes used in this study are the summation, subtraction, multiplication, protected division (returns 1 in case of division by zero), and the unary positive operator (returns zero if the provided value was negative, otherwise it returns the original value). The terminal and function nodes were selected based on previous studies [31].

C. Lexicase Selection

Lexicase selection is a selection technique designed to select individuals that perform well on part of the problem rather than individuals that perform well overall [28]. The idea is that the selected individuals will be able to pass on the traits that make them suitable for solving specific parts of the problem while becoming better in general.

Lexicase selection first randomly permutates the set of test cases from the training set and then each individual is evaluated on the first test case. The individuals that proceed to the next iteration are only those whose error is equal to the minimal error achieved by any candidate individual on this test case. The algorithm then proceeds to the next test case and repeats the same process until only one individual is left. If all test cases have been evaluated and there are still multiple candidates left, one of those individuals is randomly selected and returned. As an extension, ϵ-lexicase adjusts the condition for surviving into the next iteration by using a parameter ϵ and allowing individuals whose error is within ϵ of the minimal error to also survive [29]. In the experiments in this paper, ϵ-lexicase with an adaptive ϵ parameter was used.

The first part of calculating the parameter ϵ for a specific test case t, as shown in (1), is computing the median of errors $e_{k,t}$ where k represents a specific individual.

$$med_t = median_k(e_{k,t}) \tag{1}$$

Next, med_t is subtracted from the error $e_{k,t}$ of each individual, and the absolute value of the difference is computed as shown in (2).

$$dif_{k,t} = |e_{k,t} - med_t| \tag{2}$$

The parameter ϵ_t for test case t is then equal to the median of the differences for all individuals as denoted in (3).

$$\epsilon_t = median_k(dif_{k,t}) \tag{3}$$

In the end, individuals whose error is not within ϵ from the minimal error are filtered out. The entire procedure of the ϵ-lexicase selection procedure is presented in Algorithm 2.

III. EXPERIMENTAL RESULTS

A. Experiment Setup

In this section, we compare the performances of regular, well-known evolutionary algorithms to the performances of those same algorithms using ϵ-lexicase selection. Three versions of selection methods were tested: GP with Roulette Wheel selection, GP with Steady State Tournament (SST) selection and GP with elimination (generation gap) selection.

Roulette Wheel (RW) selection uses a fitness-proportional selection operator to select individuals which will proceed into the next generation. The selection pressure was set to 10 (i.e. the best individual has the probability of selection 10 times greater than the worst one). Crossover is then performed on randomly selected individuals in the new population. The number of crossovers performed is determined by the chosen crossover rate. The individuals in the new population are then mutated with the defined individual mutation probability. Lexicase selection was added to Roulette Wheel (denoted as RW-LEXI) by replacing fitness proportional selection. Therefore, individuals for the next population are selected using lexicase and are then randomly selected for crossover. Each algorithm was tested with the following values of the crossover probability: 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9.

Algorithm 2: ϵ-lexicase selection

```
candidates ← population
cases ← random_permutation(test_cases)
for i = 0; i < size(cases); i++ do
    case ← cases[i]
    if size(candidates) == 1 then
        return candidates
    end if
    case ← cases[i]
    for candidate ∈ candidates do
        errors ← error(candidate, case)
    end for
    min_error ← min(errors)
    for error ∈ errors do
        error ← abs(error − median(errors))
    end for
    MAD ← median(errors)
    candidates_next ← {}
    for candidate ∈ candidates do
        if error(candidate, case) ≤ MAD + min_error
        then
            candidates_next ← candidate
        end if
    end for
    candidates ← candidates_next
end for
```

Steady State Tournament selection randomly selects n individuals and then replaces the worst individual from the tournament with the child generated by performing crossover on two randomly selected surviving individuals from the tournament. This variant was called SST-RAND. The alternative, where the *best* two individuals from the tournament are selected for crossover, was also tested and was called SST-BEST.

Lexicase selection was added to SST selection in two different ways: the first modification affects the selection of individuals to participate in the tournament; the second one affects the choice of parents among the tournament survivors. In cases where lexicase was used for selection for the tournament, the corresponding variants are denoted SST-LEXI-*. For parent selection from the tournament survivors, both random selection (SST-LEXI-RAND) and selection of best individuals (SST-LEXI-BEST) was tested. On the other hand, if lexicase was used only to select individuals for crossover parents from the tournament, while the individuals for the tournament were still randomly selected, this is denoted with SST-RAND-LEXI. Each variant was tested with the following values of the tournament size: 4, 5, 6, 7, 8, 9, 10, 12, 15 and 20.

The elimination GA uses an inverse fitness-proportional elimination operator which selects individuals from the population for elimination (when an individual is selected for elimination, it cannot be selected again). Crossover is then performed on randomly selected individuals from the remaining individuals in the population. The number of eliminated individuals is determined by the generation gap parameter. Selection pressure was set to 10. In the lexicase version of the algorithm (EA-LEXI), lexicase was used instead of random selection for choosing parents for crossover. Each algorithm was tested with the following values of the generation gap parameter: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8 and 0.9.

Mutation probability of an individual was set to 0.3 for all algorithms. Population size was set to 1 000 in all experiments. The tree genotype was used in all algorithms and the maximal tree depth was set to 5. The termination criterion was set to 40 000 function evaluations.

To train GP and test the performance of the generated rules, a training and test set were defined. Both sets consist of 60 problem instances of various sizes and complexities. GP uses the training set to construct new DRs, whereas their performance is reported on the test set. The fitness of the individuals is calculated as the sum of TWT values for each of the problem instances on which they are evaluated. The values of each problem instance are normalised to ensure that all instances have a similar influence on the total fitness. Additional information about the design of the instances can be obtained from [31]. Each experiment is run 30 times to obtain significant results. From each algorithm run only the best individual on the training set is selected and evaluated on the test set. In the results the minimum, median, and maximum values of the 30 obtained solutions are reported. The Mann-Whitney statistical test is used to perform a pairwise comparison between experiments to determine whether a statistically significant difference exists. The results are considered significant if a p value less than 0.05 is obtained.

B. Roulette Wheel Genetic Algorithm

In this section, results for the Roulette Wheel GA and Roulette Wheel GA with lexicase selection are compared. Based on the results shown in Table II it can be seen that the lexicase version of the algorithm achieves a significant improvement over the classical one. For each tested parameter value, RW-LEXI outperforms RW based on the obtained medians. The Mann-Whitney statistical test demonstrated that in all cases RW-LEXI achieved significantly better results than RW for the same parameter values. Even when comparing the results obtained with the best parameters for each algorithm (denoted in boldface), lexicase is superior by all measures. Aside from the better median values that are achieved by RW-LEXI, the table also demonstrates that the results are less dispersed when using this selection variant instead of RW, which can be seen from the smaller interval between the minimum and maximum values. It is interesting to observe that both methods obtained their best results with a different crossover rate. The RW performed better when using larger values for the crossover parameter, which means that a larger part of the population will be replaced at each iteration. This can be attributed to the fact that the RW selection uses the entire problem set to determine which solutions to select, as this allows the algorithm to produce more offspring from better

solutions. On the other hand, RW-LEXI performed best for the smallest tested crossover rate. This is due to the fact that solutions are evaluated only on subsets of problem instances and thus many solutions can perform well or be the best depending on the instances that were selected. Therefore, it makes more sense to generate a smaller number of offspring to ensure diversity so that the population does not get crowded with offspring of similar individuals.

C. Steady State Tournament Genetic Algorithm

The results for the Steady State Tournament GA are shown in Table III. It is obvious that SST-RAND yielded the best results, while SST-BEST achieved slightly worse results. Out of the three lexicase versions of the algorithm, SST-RAND-LEXI achieved the best results. This could be due to the fact that lexicase selection on average selects overall better individuals than random selection which increases the selection pressure and leads to too fast convergence. For some parameters, the aforementioned algorithm even achieved better results than SST-BEST. By comparing SST-LEXI-BEST and SST-LEXI-RAND it cannot be concluded which is better since for 5 parameter values SST-LEXI-BEST is better and for the other 5 SST-LEXI-RAND is better, while for tournament size 3 the results are the same. This happens because for tournament size 3 the worst individual is eliminated and the remaining two are chosen for crossover, regardless of whether random or best selection is used. The same goes for SST-RAND and SST-BEST when using tournament size 3. By comparing the best results from the lexicase algorithms, it is clear that SST-RAND-LEXI has a significantly smaller range from the minimum to the maximum value than the other two algorithms. This difference is likely caused by using different selections for the tournament, which greatly impacts the dispersion of the quality of individuals in the new population. The same is true for the difference in dispersion between the regular SST algorithms (SST-RAND, SST-BEST) and lexicase versions (SST-LEXI-BEST, SST-LEXI-RAND). It should be observed that SST-LEXI-BEST, SST-LEXI-RAND and SST-RAND-LEXI all achieve their best results for tournament sizes greater than 10, while SST-RAND and SST-BEST performed better with smaller tournament sizes. Since lexicase selects diverse solutions which are suitable for solving a specific part of the problem, it is desirable to select more individuals for the tournament to ensure that more parts of the solution are covered in the selected tournament.

D. Elimination Algorithm

The results for the Elimination GA are shown in IV. It is immediately visible that the best parameters are different for EA and EA-LEXI. Although EA-LEXI achieves better results for most parameters, according to the Mann-Whitney statistical test the difference is only significant for generation gap of 0.1, 0.5 and 0.6. However, there is no statistically significant difference between the best results from both algorithms. Nevertheless, it can be concluded that introducing lexicase to EA in some cases does improve the algorithm's performance. It is interesting to observe that in this case the dispersion of the results is similar in both tested variants. This makes sense since, in EA-LEXI, lexicase selection is only used to select parents for crossover while individuals for the next generation are still selected based on their overall fitness which diminishes the diversity of solutions that lexicase selection usually introduces. This could also explain why the improvements of using lexicase selection in EA are not as big as in e.g. Roulette Wheel GA.

IV. Discussion

Table V shows the best results achieved by each tested algorithm. The same results are additionally displayed as a box plot in Figure 1. Overall, when comparing by the median, the three algorithms that stand out with the best results are RW-LEXI, SST-RAND and SST-RAND-LEXI. Although RW-LEXI achieved the best overall median value, the results it obtained are not significantly different from the other two algorithms. Since SST-RAND achieved the best minimum and maximum values and its median is very close to the best value achieved by RW-LEXI, it could be concluded that overall SST-RAND was the best performing algorithm. Additionally, the box plot shows that the best three algorithms have the least dispersed results, which is beneficial as it denotes that they are more likely to obtain better solutions.

Figure 2 shows the convergence of the best selection methods on the train and test sets for the median value obtained based on 30 executions. Convergence on the test set was obtained by evaluating the best individual from training after a certain number of evaluations on the test set. The convergence graph on the test set demonstrates that all algorithm variants that use lexicase selection converge much faster to better solutions. The only exception here is for the elimination selection, where the standard variant has a faster convergence rate at the start but then easily gets stuck in local optima. On the other hand the EA-LEXI does not have such a problem and easily converges to a better solution. However, more important than the convergence on the training set is the convergence behaviour on the test set. Here the superiority of lexicase based selections can easily be observed. In the first quarter of the algorithm run time, the lexicase based selections converge quite quickly to good solutions, whereas their standard counterparts need much more time to obtain solutions of the same fitness level. This can be best observed for SST where the variant that uses lexicase for selection converges to a median of 14 in only 3 000 evaluations. The standard SST variant of reached the same value after 12 000 evaluations. This demonstrates that by using the lexicase selection good solutions can be achieved in only a fraction of time in comparison to the standard selection.

One important thing which needs to be outlined for the lexicase based selection algorithms is that in our experiments the evaluation of the individuals was always done on the entire training set. Regardless of that, during the selection process only the fitness values of the chosen instances were used to select the parents. An alternative approach would be that only those instances that are used for selection are

TABLE II
ROULETTE WHEEL GA RESULTS

Crossover rate		0.3	0.4	0.5	0.6	0.7	0.8	0.9
RW	**min**	13,404	13,424	13,338	**13,371**	13,404	13,364	13,278
	med	14,708	14,708	14,725	**14,491**	14,600	14,581	14,947
	max	16,629	18,621	17,702	**17,290**	16,481	19,383	18,588
RW-LEXI	**min**	**12,976**	12,316	12,928	12,821	12,405	12,477	12,370
	med	**13,646**	13,881	13,909	13,946	13,792	14,016	13,831
	max	**15,664**	14,822	16,185	15,691	15,400	15,815	15,249

TABLE III
STEADY STATE TOURNAMENT GA RESULTS

Tournament size		3	4	5	6	7	8	9	10	12	15	20
SST-RAND	**min**	13,038	13,012	12,901	**12,899**	13,210	13,058	13,181	13,117	12,895	13,063	12,960
	med	13,740	13,835	13,808	**13,659**	14,170	13,755	13,797	13,748	13,659	13,733	13,830
	max	14,683	15,473	15,042	**14,935**	15,165	14,977	14,983	15,203	15,442	15,698	15,195
SST-BEST	**min**	13,038	13,100	13,071	13,089	13,082	13,059	**13,012**	13,254	13,097	13,173	12,938
	med	13,740	14,132	13,897	14,261	13,869	14,142	**13,830**	14,085	13,950	14,151	14,312
	max	14,683	14,876	14,973	15,042	16,360	16,503	**15,671**	15,311	16,053	15,074	15,370
SST-LEXI-BEST	**min**	14,203	13,227	13,580	13,896	13,673	13,371	13,696	13,427	13,511	**13,535**	13,777
	med	17,898	17,132	15,963	16,591	16,153	14,926	15,396	15,593	16,396	**14,875**	15,173
	max	27,654	25,337	21,625	20,012	23,418	18,537	20,150	19,990	20,732	**19,360**	20,626
SST-LEXI-RAND	**min**	14,203	14,141	13,661	13,225	13,295	13,526	13,389	13,580	13,580	13,420	**13,443**
	med	17,898	16,744	16,787	14,962	15,429	15,690	15,754	15,817	15,811	14,923	**14,598**
	max	27,654	21,092	24,740	20,501	20,644	21,339	19,569	23,553	19,730	19,219	**18,741**
SST-RAND-LEXI	**min**	13,020	13,079	13,262	12,751	12,962	13,011	13,251	12,928	**13,177**	12,811	12,901
	med	14,001	14,064	13,863	13,862	13,763	13,835	13,832	13,965	**13,669**	13,831	13,799
	max	17,489	15,172	14,978	15,048	15,583	15,028	15,190	15,346	**15,167**	14,723	15,449

TABLE IV
ELIMINATION ALGORITHM RESULTS

Generation gap		0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
EA	**min**	14,153	13,871	13,638	**13,923**	13,546	13,869	13,404	13,610	13,787
	med	16,594	15,972	16,089	**15,393**	16,120	16,2659	15,576	15,555	15,471
	max	19,078	18,631	18,908	**19,653**	19,720	18,8615	19,819	18,643	20,612
EA-LEXI	**min**	13,798	14,086	13,664	13,693	13,393	**13,550**	13,510	12,694	13,923
	med	15,652	15,402	15,304	15,515	15,023	**14,727**	15,114	15,039	15,663
	max	18,864	20,236	21,987	19,934	18,280	**19,389**	18,588	18,633	20,815

TABLE V
BEST RESULTS FOR ALL ALGORITHMS

	min	med	max
RW	13,371	14,491	17,290
RW-LEXI	12,976	**13,646**	15,664
EA	13,923	15,393	19,653
EA-LEXI	13,550	14,727	19,389
SST-RAND	**12,899**	13,659	**14,935**
SST-BEST	13,012	13,830	15,671
SST-LEXI-BEST	13,535	14,875	19,360
SST-LEXI-RANDOM	13,443	14,598	18,741
SST-RAND-LEXI	13,177	13,669	15,167

evaluated and contribute to individual fitness. However, this may lead to a poor performance of the entire algorithm as it would not use elitism anymore. This is due to the fact that if individuals are evaluated only on subsets of instances, then it is possible that the DR that generally performs best is eliminated because other rules perform better on that subset of instances. In such conditions significantly worse performances were achieved. Therefore it was decided to evaluate solutions on the entire problem set so that the best overall solutions are not eliminated. However, this means that lexicase selection will have the same execution time as other selection types, although it uses fewer instances to perform decisions.

V. CONCLUSION

In recent years several studies have focused on improving the performance of the hyper-heuristic methods for scheduling problems by putting more emphasis on the problem sets that

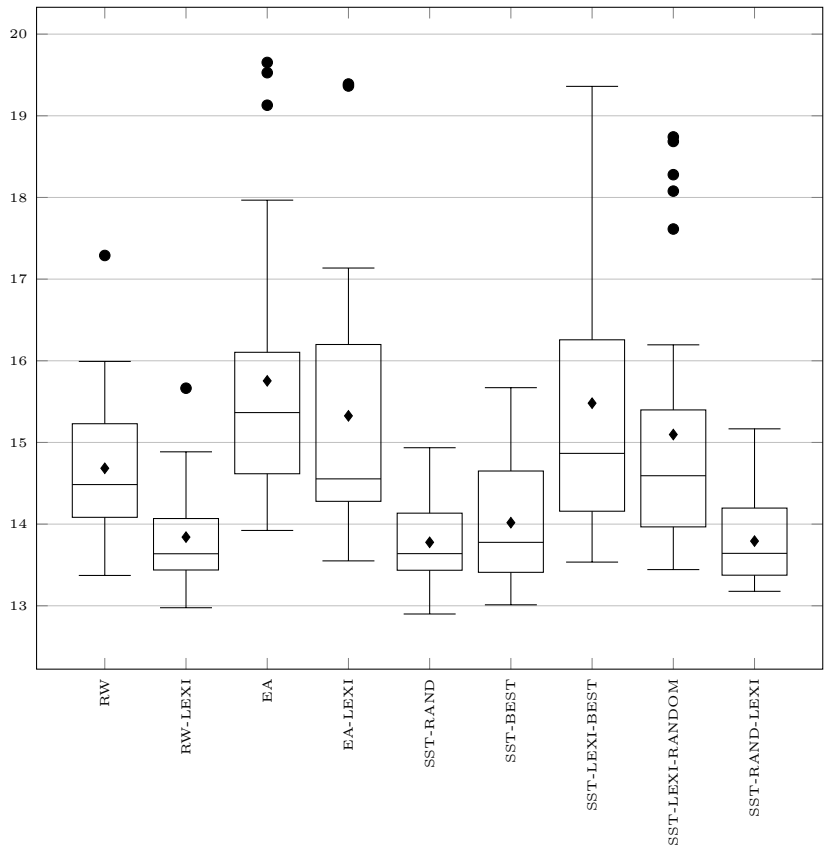
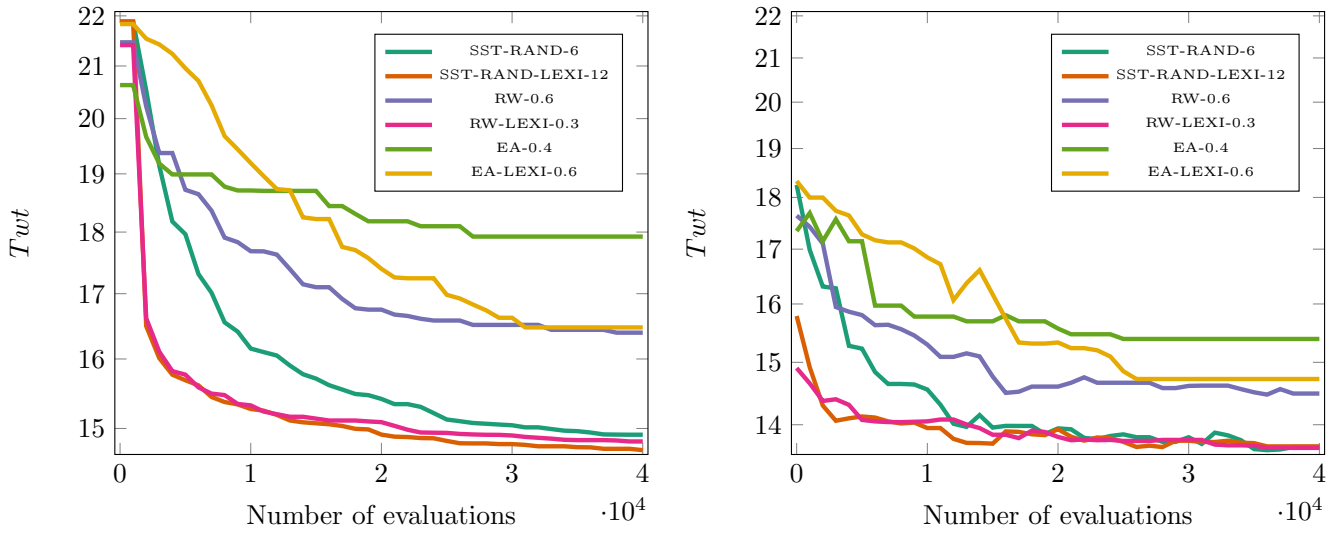


Fig. 1. Box plot of the results obtained with the best parameter values for each tested algorithm



(a) Train set

(b) Test set

Fig. 2. Algorithm convergence comparison

are used during the learning process. These studies already demonstrated that the performance of automatically generated DRs could be improved by trying to increase the diversity of the population and by training the solutions on those instances which are most informative. Based on that idea, this paper proposes the application of the lexicase selection in GP to improve the diversity of the population and consequentially improve the quality of the generated rules.

The obtained results demonstrate that applying the lexicase

selection in the RW and elimination based algorithms leads to better results in all cases. For the SST algorithm variant, the lexicase selection was unable to significantly improve the results. However, using lexicase usually results in a faster convergence, which means that better solutions can be achieved in a smaller amount of time. Therefore, it can be concluded that the additional diversity provided by the lexicase selection has a positive effect on the evolution process, both in terms of convergence and solution quality.

Since the application of lexicase demonstrated promising results, there are several possibilities for the extension of this work. One direction would be aimed at improving the run time of lexicase, since it was already outlined that individuals are evolved on the entire test set. Additionally, the current lexicase selection selects the problem instances completely randomly; it might make sense to investigate if different strategies of selecting problem instances could improve the overall performance. Finally, another line of research that could be used is to combine surrogate models with lexicase selection to obtain benefits of both methods.

REFERENCES

[1] J. A. Castillo-Salazar, D. Landa-Silva, and R. Qu, "Workforce scheduling and routing problems: literature survey and computational study," *Annals of Operations Research*, vol. 239, no. 1, pp. 39–67, Aug. 2014. [Online]. Available: https://doi.org/10.1007/s10479-014-1687-2

[2] M. Kofler, S. Wagner, A. Beham, G. Kronberger, and M. Affenzeller, "Priority Rule Generation with a Genetic Algorithm to Minimize Sequence Dependent Setup Costs," in *Computer Aided Systems Theory - EUROCAST 2009: 12th International Conference, Las Palmas de Gran Canaria, Spain*. Springer Berlin Heidelberg, 2009, pp. 817–824.

[3] E. Burke, P. De Causmaecker, G. Vanden Berghe, and H. Van Landeghem, "The state of the art of nurse rostering," *J. Scheduling*, vol. 7, pp. 441–499, 11 2004.

[4] M. L. Pinedo, *Scheduling: Theory, algorithms, and systems: Fourth edition*. Boston, MA: Springer US, 2012, vol. 9781461423614.

[5] E. Hart, P. Ross, and D. Corne, "Evolutionary Scheduling: A Review," *Genetic Programming and Evolvable Machines*, vol. 6, no. 2, pp. 191–220, Jun. 2005.

[6] S. Nguyen, D. Thiruvady, A. T. Ernst, and D. Alahakoon, "A hybrid differential evolution algorithm with column generation for resource constrained job scheduling," *Computers and Operations Research*, vol. 109, pp. 273 – 287, 2019.

[7] I. Vlašić, M. Đurasević, and D. Jakobović, "A comparative study of solution representations for the unrelated machines environment," *Computers & Operations Research*, vol. 123, p. 105005, 2020.

[8] 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.

[9] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 107–131, Nov. 1999.

[10] M. Đurasević and D. Jakobović, "A survey of dispatching rules for the dynamic unrelated machines environment," *Expert Systems with Applications*, vol. 113, pp. 555 – 569, 2018.

[11] D. Jakobović and K. Marasović, "Evolving priority scheduling heuristics with genetic programming," *Applied Soft Computing*, vol. 12, no. 9, pp. 2781–2789, Sep. 2012.

[12] 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.

[13] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: a survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, Dec. 2013. [Online]. Available: http://link.springer.com/10.1057/jors.2013.71

[14] 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.

[15] J. Branke, S. Nguyen, C. W. Pickardt, and M. Zhang, "Automated Design of Production Scheduling Heuristics: A Review," *IEEE Transactions on Evolutionary Computation*, vol. 20, no. 1, pp. 110–124, Feb. 2016.

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

[17] 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

[18] 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.

[19] 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

[20] 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

[21] 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.

[22] 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.

[23] T. Hildebrandt, J. Heger, and B. Scholz-Reiter, "Towards improved dispatching rules for complex shop floor scenarios," in *Proceedings of the 12th annual conference on Genetic and evolutionary computation - GECCO '10*. New York, New York, USA: ACM Press, 2010, p. 257.

[24] S. Nguyen, M. Zhang, and K. C. Tan, "Surrogate-assisted genetic programming with simplified models for automated design of dispatching rules," *IEEE Transactions on Cybernetics*, vol. 47, no. 9, pp. 2951–2965, 2017.

[25] F. Zhang, Y. Mei, S. Nguyen, and M. Zhang, "Collaborative multifidelity-based surrogate models for genetic programming in dynamic flexible job shop scheduling," *IEEE Transactions on Cybernetics*, pp. 1–15, 2021.

[26] D. Karunakaran, Y. Mei, G. Chen, and M. Zhang, "Active sampling for dynamic job shop scheduling using genetic programming," in *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2019, pp. 434–441.

[27] D. Karunakaran, Y. Mei, G. Chen, and M. Zhang, "Sampling heuristics for multi-objective dynamic job shop scheduling using island based parallel genetic programming," in *Parallel Problem Solving from Nature – PPSN XV*. Springer International Publishing, 2018, pp. 347–359.

[28] L. Spector, "Assessment of problem modality by differential performance of lexicase selection in genetic programming: A preliminary report," in *Proceedings of the 14th Annual Conference Companion on Genetic and Evolutionary Computation*, ser. GECCO '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 401–408. [Online]. Available: https://doi.org/10.1145/2330784.2330846

[29] W. La Cava, L. Spector, and K. Danai, "Epsilon-lexicase selection for regression," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 741–748.

[30] 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.

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