Applied Intelligence
https://doi.org/10.1007/s10489-021-02227-3

Genetic programming hyperheuristic parameter configuration using fitness landscape analysis

Rebeka Čorić[1] · Mateja Đumić[1] · Domagoj Jakobović[2]

Accepted: 22 January 2021

Abstract
Fitness landscape analysis is a tool that can help us gain insight into a problem, determine how hard it is to solve a problem using a given algorithm, choose an algorithm for solving a given problem, or choose good algorithm parameters for solving the problem. In this paper, fitness landscape analysis of hyperheuristics is used for clustering instances of three scheduling problems. After that, good parameters for tree-based genetic programming that can solve a given scheduling problem are calculated automatically for every cluster. Additionally, we introduce tree editing operators which help in the calculation of fitness landscape features in tree based genetic programming. A heuristic is proposed based on introduced operators, and it calculates the distance between any two trees. The results show that the proposed approach can obtain parameters that offer better performance compared to manual parameter selection.

Keywords Fitness landscape analysis · Genetic programming · Scheduling · Tree operators · Clustering · Parameter configuration

Article Highlights

- Hyperheuristic fitness landscape with tree representation is examined.
- Operators that help obtain random walk and a heuristic for calculating the distance between any two trees are introduced.
- Clustering similar instances based on fitness landscape features leads to better parameters for genetic programming.

✉ Rebeka Čorić
rcoric@mathos.hr

Mateja Đumić
mdjumic@mathos.hr

Domagoj Jakobović
domagoj.jakobovic@fer.hr

[1] Department of Mathematics, J.J. Strossmayer University of Osijek, Gajev trg 6, 31000 Osijek, Croatia

[2] Faculty of Electrical Engineering and Computing, Unska 3, 10000 Zagreb, Croatia

1 Introduction

Fitness landscape analysis is a tool for gaining insight into a problem structure and acquiring more information about the problem we are trying to solve. It can be used for various purposes, such as acquiring a better understanding of combinatorial optimization problems, determining the degree of difficulty of solving a problem using a certain algorithm, determining the effectiveness of the use of certain algorithms for solving a given problem, identifying the algorithm or algorithm parameters that are best suited for solving a given problem, etc. In this paper, the main idea is to use fitness landscape analysis in order to identify good parameters for genetic programming (GP) (with individuals represented as trees) that can solve the scheduling problems.

GP is an evolutionary algorithm inspired by the biological evolution proposed by Koza in 1992 [29]. GP aims to find computer programs that perform a user-defined computational task rather than providing a solution to the task itself. It has been widely used in many different areas throughout the years after its introduction. Its applications are explored in some of the newer papers that range from those on medicine [48], mechanics [16, 18], hydrology

Published online: 09 March 2021

[31, 34], and forecasting storm surges [21] to those on economy [25], customer satisfaction prediction [7], home energy-management systems [56], music [9], psychology [1], and many other areas. In addition to its various applications, the more recent papers also investigate the theoretical aspects of GP. The authors in [50] propose two approaches for reducing code bloat in GP, while the authors in [46] evolve dynamic fitness measures for GP. A body of work also adresses GP for symbolic regression, from analyzing and applying it in order to improve the results in symbolic regression [19] to predicting GP performance [2] and proposing frameworks for solving symbolic regression problems by using GP [10]. The authors in [51] describe how a grammar-guided GP model was used and defined as a free-parameter algorithm that self-adapts to the required parameters and searches for highly representative continuous patterns. This work uses GP for parameter tuning, but it does not use fitness landscape analysis to do so.

Scheduling is a process of allocating resources to tasks over a period of time with a goal to optimize one or more objectives [42]. It plays an important role in manufacturing and production systems, information processing environments, transportation, and many other areas; therefore, it is widely explored by researchers. The process of scheduling involves a certain number of jobs that need to be scheduled on one or more machines while accounting for constraints. Based on the number of machines and the types of constraints, scheduling problems can be divided into different categories. Some of the scheduling environments are the following: single machine environment, identical machines in parallel, unrelated machines in parallel, flow shop, job shop, etc. Some of the constraints that can appear in the aforementioned environments are the following: release dates of jobs, preemptions, precedence constraints, breakdowns, etc. Further information about environments and constraints can be found in [42]. Every scheduling task needs to include one or more objective functions that must be optimized. The most common objective functions are the following: makespan, total weighted completion time, total weighted tardiness, weighted number of tardy jobs, etc. By combining different environments, constraints, and objective functions, one can describe almost every scheduling problem that arises in real life. Therefore, it is not surprising that researchers continue to work in this area actively. Some of the newer papers focus on analyzing ideal schedules for parallel machine environment [23], proposing methods for scheduling in flexible job shop environments [6], comparing schedule generation schemes for unrelated machines environment [15], studying single machine environments with different constraints [55], etc.

In this paper, a single machine environment, an unrelated machine environment, and RCPSP will be used as underlying problems for the proposed approach toward obtaining good parameters for GP. These three problems will be described in Section 2.

The previous work in the area of fitness landscape analysis has primarily focused on determining the fitness landscape features for heuristics and not hyperheuristics. Ochoa et al. defined hyperheuristic search space for production scheduling and timetabling problems in [40] and [39]; however, in those papers, individuals are represented in binary and permutation representation. The question that arises, then, is how to carry out fitness landscape analysis on tree individuals. Some tree edit distance functions are available for measuring the distance between two trees [33, 41], but those distance functions do not take into account the fact that in cases of genetic programming where an individual is represented as a tree, the nodes can be either function or terminal nodes and function nodes can have a varying number of child nodes. In this paper, dedicated tree operators will be proposed, and they will enable us to calculate the distance between any two trees while taking into account those different nodes. In [30], Liefooghe et al. used fitness landscape features for sorting out instances of the NK model [24] into clusters and then they determined optimal parameters for the given metaheuristic in every cluster. They only used three fitness landscape features: autocorrelation coefficient, average fitness, and neutrality. This paper draws on this idea and expands it. The contributions of this paper are the following:

- instead of the metaheuristic landscape, the hyperheuristic landscape is examined;
- new operators are introduced in order to obtain random walk on an expression tree representation;
- a heuristic for calculating the distance between any two trees is presented;
- multiple fitness landscape features are taken into account.

As mentioned previously, hyperheuristic fitness landscapes are not as widely examined as metaheuristics fitness landscapes. Ochoa et al. stated that in a hyperheuristic fitness landscape, one has to deal with two search spaces: the search space of heuristics and the optimization problem solution space; however, there is a single landscape that is obtained after evaluating a point from the heuristic search space in the problem solution space [39]. In this paper, the fitness landscape of genetic programming is examined and used, following the principle described in the aforementioned paper; however, tree representation is used here and fitness landscape features are calculated for tree individuals. In order to obtain the fitness landscape features of the GP fitness landscape, new operators are introduced, which enable us to identify the neighbors of a given tree, thereby obtaining the random walk required for calculating some of the features. Additionally, in order to calculate some of the features, the distance between

any two trees is required; therefore, the heuristic that uses the introduced operators to calculate the distance between trees is presented and explained. Additionally, with three fitness landscape features already mentioned and used in [30], in this paper, five other fitness landscape measures are examined for the GP fitness landscape of scheduling problems. The paper is organized as described here. Section 2 briefly describes the following scheduling problems: single machine environment, unrelated machines environment, and resource constrained project scheduling problem (RCPSP), which are used in this paper, as well as GP. In Section 3, the fitness landscape analysis is described and a list of fitness landscape features that are calculated for the aforementioned scheduling problems are given. Additionally, feature selection of those fitness landscape features is described. Section 4 introduces tree operators that make it possible to obtain random walks and calculate fitness landscape features. Moreover, a heuristic that calculates the distance between any two trees based on the introduced operators is given in the same section. In Section 5, problem instances are given and the clustering procedure is explained. Furthermore, this section explains how the parameter configurations for the clusters are obtained, and the results are shown. Finally, Section 6 concludes the paper.

2 Scheduling problems and genetic programming

In this paper, the problems that were investigated are single machine scheduling, unrelated machine scheduling, and resource constrained project scheduling problem (RCPSP). Each of them is briefly described below. GP is also briefly explained in the second part of this section.

2.1 Scheduling problems

In scheduling, the single machine environment is very simple, and it represents a special case of all other environments [42]. This environment only contains one machine on which all the activities must be scheduled with various possible restrictions and constraints, and many possible objective functions. In this paper, single machine environment was used with the following assumptions: all of the information about jobs is known in advance, scheduling is done online, no preemption is allowed, and the machine is always available (there are no breakdowns). The objective function that was minimized indicates the total weighted tardiness.

In order to schedule the jobs on the machine, the following schedule generation scheme was used (as suggested in [22]): as long as unscheduled jobs exist, wait until the machine becomes available, calculate the priorities of all unscheduled jobs, choose the job with the best priority, and schedule it on the machine. A schedule was obtained in this way, and it was evaluated in order to derive a value (fitness) of the objective function.

In an unrelated machines environment, m machines are available in parallel. For every machine and every job, there is a certain speed at which a machine can process a job. [42] The goal is to assign jobs to machines and optimize the criterion function. In this environment, the following assumptions were used: not all information about jobs is known in advance, jobs become available for scheduling at some point of time and, therefore, online scheduling is used, no preemption is allowed, and all the machines are always available (there are no breakdowns). As in the case of the single machine environment, the total weighted tardiness was used as the objective function here, and the goal was to minimize it.

As with the single machine environment, it was necessary to define the schedule generation scheme. The following procedure was used (as suggested in [13]): while unscheduled jobs exist, wait until at least one job and one machine are available. Calculate the priority values for each available unscheduled job on every machine. Based on priority value, the best machine for every available unscheduled job should be determined. Out of all jobs for which the best machine is available, the one with the highest priority for scheduling is to be selected. If there are no jobs for which the best machine is available, scheduling is to be postponed until another job or another machine becomes available.

RCPSP is an NP-hard scheduling problem that involves n activities with known durations, certain precedence constraints for those activities, and m resources for which the capacity is constrained. Every activity needs to avail a certain amount of given resources in order to be completed. The goal is to find a schedule so that the precedence constraints and constraints on resource amount are met and the chosen optimization criteria are minimized or maximized (based on the nature of the chosen criterion). In this paper, the normalized makespan was given as $f_i = \frac{C_i}{p_i^{avg} \cdot \sqrt{n_i}}$ (C_i indicates the achieved makespan, p_i^{avg} indicates the average duration of the activity, and n_i indicates the number of activities for project instance i), which is defined in [14] and was chosen as the optimization criterion and minimized. The number of activities n_i was used in this formula because RCPSP can have a different number of activities in multiple problem instances, and this function takes that possibility into account and balances the end result. A formal definition of RCPSP can be found in [3].

While solving RCPSP, one can use exact methods or heuristic methods. As RCPSP is NP-hard, it is impossible to solve it exactly in a reasonable amount of time in cases where there is a large number of activities. Many heuristic

approaches have been developed for solving this problem, and they can be classified into two categories: priority rules-based methods (constructive heuristics) or metaheuristic-based approaches (improvement heuristics) [27]. The first category (which was used here) is based on the construction of a complete schedule (starting from an empty schedule) using a schedule generation scheme [28] and priority rules [27]. The schedule generation scheme starts with an empty schedule and builds and updates partial schedules until all activities are scheduled, and during that process, it takes into account the precedence and resource constraints. When choosing the activity that is to be scheduled next, the schedule generation scheme looks at the priorities assigned to the activities by using the priority rule and chooses the activity with the minimum or maximum priority (based on the nature of the rule). Schedule generation schemes can be serial or parallel, and they differ in regard to the way in which activities and time slots are handled during the scheduling procedure. Here, a parallel schedule generation scheme was used. More details about schedule generation schemes as well as a description of the parallel schedule generation scheme can be found in [28].

2.2 Genetic programming

GP is an evolutionary algorithm inspired by the biological evolution, and it aims to find computer programs that perform a user-defined computational task rather than providing a solution to the task itself. The main advantage of GP is that it is able to evolve higher-level programs that can be used to generate solutions to a variety of problem instances. GP is used in a variety of scheduling problems, such as single machine scheduling [12], job shop scheduling [38], unrelated machines scheduling [13], RCPSP [14], etc. In the case of scheduling problems, it is used to evolve priority rules that the schedule generation scheme will take into account while creating a feasible schedule. GP is mostly used in dynamic environments because the rule that is once evolved can be used for different problem instances and can provide results in a short time as opposed to metaheuristic approaches. Additionally, GP can react to unforeseen changes that happen dynamically in such systems. In order to be able to make useful priority rules, GP needs to have a set of functions and terminals from which it can choose building blocks for priority rules. The set of functions that was used can be seen in Table 1. It was up to GP to use these simple functions in order to create more complex operators. For RCPSP, all of the functions from Table 1 were used, while in the case of a single machine and unrelated machine environments, only addition, subtraction, multiplication, protected division, and the POS function were used.

Table 1 Subset of functions for genetic programming

Function	Description
+, -, *	addition, subtraction and multiplication
/	Protected division: $DIV(a,b) = \begin{cases} 1, & \lvert b\rvert < 0.000000001 \\ \frac{a}{b}, & otherwise. \end{cases}$
MAX	$MAX(a) = \begin{cases} a, & a > 0 \\ 0, & otherwise. \end{cases}$
POS	$POS(a) = \begin{cases} a, & a > 0 \\ -a, & otherwise. \end{cases}$
NEG	$NEG(a) = -a$
IF	$IF(a,b,c) = \begin{cases} b, & a > 0 \\ c, & otherwise. \end{cases}$

In regard to terminal set, it is always advisable to chose domain-specific terminals. In this paper, different terminals were used for different scheduling problems.

Table 2 depicts the terminals for the single machine scheduling environment. The terminals pt, dd, and w provide relevant information about the jobs that need to be scheduled, so it is logical to include them in the terminal set. The terminals SP and SD also provide useful information about the state of the system. The terminals Nr and SPr were added to the terminal set because in every step, priorities were recalculated; therefore, the history of system work did not have an impact on the decision about the next job. In order to acquire information about the urgency of scheduling a job, SL was added, which provides information about the length of time for which the scheduling of a job can be postponed while ensuring that the due date is not exceeded.

Table 3 shows the terminals for the unrelated machines scheduling environment. The first four terminals are the same as in the single machine environment because they are related to jobs that have the same properties in both the single machine as well as the unrelated machines environment. The unrelated machines environment includes

Table 2 Terminals for genetic programming for the single machine scheduling environment

Terminal	Description
pt	processing time of job j on the machine i (p_{ij})
dd	due date (d_j)
w	weight (w_j)
N	total number of jobs
Nr	number of unscheduled jobs
SP	sum of job processing times
SPr	sum of job processing times for unscheduled jobs
SD	sum of due dates of all jobs
SL	positive slack ($max\{d_j - p_j - time, 0\}$)

Table 3 Terminals for genetic programming for the unrelated machines scheduling environment

Terminal	Description
pt	processing time of job j on the machine i (p_{ij})
dd	due date (d_j)
w	weight (w_j)
SL	positive slack ($max\{d_j - p_j - time, 0\}$)
pmin	the minimum job processing time on all machines: $min_i(p_{ij})$
pavg	the average processing time on all machines
PAT	patience
MR	machine ready
age	the time that the job spent in the system: $time - r_j$

more machines, so the information about the processing times on different machines ($pmin$ and $pavg$) can be useful while building an individual for GP. The terminal PAT provides information about the amount of time that will be needed for the machine with the minimum processing time for the current job to become available, while the terminal MR provides the information about the amount of time that will be needed for the current machine to become available. Finally, the terminal age indicates the time spent by the job in the system before the system can determine which job to schedule next so that no job has to wait for too long.

The terminals that were used for RCPSP are shown in Table 4. The terminals were divided into project-specific and activity-specific terminals. Additionally, the terminals can be static (i.e., their value is unchanged throughout the entire execution) and dynamic (i.e., their value can change according to the current state of activities and resources). The dynamic terminals include the following: NUA, NAA, NPA, SUD, SAD, SPD, NSP, and SL, while all other terminals are static. Project-specific terminals deal with information regarding the entire project. The terminals RF and RS as well as TNA can be obtained from the configuration file for the RCPSP instance. The total project duration (TD) is obtained by summing up the processing times of all the activities. While running the program, some activities will be scheduled (i.e., processed), some will be active (i.e., available for scheduling because constraints are satisfied), and some will remain unscheduled (i.e., unprocessed). The terminals NUA, NAA, NPA, SUD, SAD, and SPD deal with the quantities and durations of those activities. Activity-specific terminals deal with information that can be obtained from a specific activity. The activity duration represents an important information in the process of determining a priority, so it was added to the terminal set. The terminals RR, RRT, and ARU deal with the resource constraints, while DPC, DSC, TPC, TSC, SPC, and SSC deal with the precedence constraints. In order to calculate a priority, it is important to know the amount of resource that an activity needs and the number of predecessors and successors that an activity has. Activities that have many successors should probably get higher priority so that its many sucessors can be scheduled earlier rather than later. The terminal $GRPW*$ provides information about the greatest positional weight rank of the activities' successors. The terminals ES, EF, LS, and LF provide information about the earliest and the latest time at which an activity can be started or finished so that one can know the timespan during which an activity must be scheduled. Activities with smaller timespans should probably get higher priority. SL has the same meaning as that in the single machine environment.

Additionally, in all three cases, the terminals 0 and 1 were added so that the operators described in Section 4 could be used.

Details about the implementation of GP for solving RCPSP, which was used for experiments here, can be found in [14]. It is also important to mention that while looking at the fitness landscape of GP, we look at the space of trees (expressions), which represent evolved priority functions, and as the fitness value of a tree we use the value of the objective function selected for the chosen problem as the fitness value of a tree, which is obtained when the schedule generation scheme makes a schedule using that evolved priority rule.

3 Fitness landscape analysis

While solving a certain problem using heuristics or hyper-heuristics such as genetic algorithm or GP, it is important to determine the appropriate algorithm parameters in order to obtain better solutions. That is where fitness landscape analysis can help. Fitness landscape was first introduced as an idea in Wright's paper [54] in 1932. Although the term "fitness landscape" is not mentioned anywhere in that paper, Wright is considered to have conceptualized that idea. The concept of fitness landscape indicates the mapping of the genome of a population of individuals and the visualization of such mapping [26]. If a problem is two-dimensional, it is easy to depict individuals on a plane and their fitness on the third axis; in this way, a landscape with peaks and valleys is obtained, and the structure of the problem can be observed and studied. However, in higher-dimensional problems, that kind of depiction cannot be made; therefore, some kind of formal definition is needed. Fitness landscape $\mathcal{F}$ can be defined as a tuple $(f, d, \mathcal{S})$ where f is a fitness function and d is a distance function between individuals which come from a given set of individuals $\mathcal{S}$ [43]. It can be seen that the fitness landscape depends not only on the fitness values of individuals but also on their position relative to each other.

Table 4 Terminals for genetic programming for RCPSP

Category	Terminal	Description
Project-specific	RF	resource factor
	RS	resource strength
	TNA	total number of activities (not including dummies)
	TD	total project duration (horizon)
	NUA	number of unprocessed activities
	NAA	number of active activities
	NPA	number of processed activities
	SUD	sum of durations of unprocessed activities
	SAD	sum of durations of active activities
	SPD	sum of durations of processed activities
Activity-specific	D	activity duration
	RR	number of required resources
	RRT	RR times quantity required for each resource
	ARU	average resource usage
	DPC	number of direct predecessors
	DSC	number of direct successors
	TPC	total number of predecessors
	TSC	total number of successors
	SPC	number of stages (levels) in predecessors' tree
	SSC	number of stages (levels) in successors' tree
	GRPW*	greatest rank positional weight all
	ES	earliest activity start
	EF	earliest activity finish
	LS	latest activity start
	LF	latest activity finish
	NSP	number of scheduled predecessors
	SL	slack: $max(EF - D - time, 0)$

Using this formal definition, many measures are developed, and they can describe the fitness ladscape of a given problem. Some of them are the following: modality [43], landscape walks (e.g., random walk, adaptive walk, uphill-downhill walk) [43], ruggedness [53], fitness distance correlation [43], neutrality [43], average fitness [30], lengthscale [37], random probing [5], information content [52], information stability [52], etc.

Fitness landscape analysis can be used in various ways. Some of them are the following: fitness landscape analysis of combinatorial optimization problems [4, 45, 49], using fitness landscape analysis to determine the degree of difficulty of solving a problem using an algorithm [26], using fitness landscape analysis to determine the effectiveness of the use of some algorithms in solving a given problem [35], using fitness landscape analysis to determine the algorithm or algorithm parameters that are best suited for solving a given problem [30, 44], etc. At the beginning of this section, it has already been stated that while solving a problem using a hyperheuristic such as GP, it is important to somehow determine the best parameters for GP in order to get better solutions. We can use measures that describe the fitness landscape of a problem to acquire a better understanding of that problem and determine the best parameters for GP that will solve our problem. Here, one should use those fitness landscape measures that can be determined relatively fast so that the algorithm parameter configuration does not take up more time than that taken to solve the problem itself.

The measures used in this paper are the following: random walk, average fitness of a random walk, autocorrelation coefficient, rate of neutral neighbours, random probing measures, information content, partial information content, information stability, and lengthscale measures. Random walk helps us to obtain some of the other measures (chapter Section 4 will explain the exact process by which random walk can be obtained for the problems given in this paper). The average fitness of a random walk is calculated as $\bar{f} = \frac{1}{l}\sum_{t=1}^{l} f(x_t)$ where $f(x_t)$ is the fitness value of a candidate solution x_t, which comes from a random walk

$\langle x_0, x_1, \ldots, x_l \rangle$ of length l. The autocorrelation coefficient is calculated as $\hat{r}(k) = \frac{\sum_{t=1}^{l-k}(f(x_t)-\bar{f})\cdot(f(x_{t+k})-\bar{f})}{\sum_{t=1}^{l}(f(x_t)-\bar{f})^2}$, and here, the autocorrelation coefficient $r(1)$ was used; the larger $r(1)$, the smoother the landscape [30]. The rate of neutral neighbours was calculated as $\frac{|NN|}{l}$ where NN is a set of neighbours in a random walk of length l, which have the same fitness value. The random probing measures include the following: random probing min, random probing max, and random probing range, and they were calculated as described in [5]; however, instead of a 100 random candidate solutions, l of them were used, where l is the length of a random walk. The information content, partial information content, and information stability were calculated as described in [52]. Finally, lengthscale was calculated as $r = \frac{|f(x_i)-f(x_j)|}{d(x_i,x_j)}$ [37] where d denotes the distance between the candidate solutions x_i and x_j. Lengthscale was calculated for every pair of candidate solutions in a random walk, and for every random walk, minimum, maximum, median, lower and upper quartile, average, and standard deviation of lengthscales were calculated in order to derive comparable measures. Additionally, the lengthscale mesures for data from which zeros are removed were calculated.

3.1 Feature selection of fitness landscape features

Feature selection approaches can be broadly categorized into four categories: filters, wrappers, hybrid, and embedded. [20] Feature selection for supervised learning is much more investigated than feature selection for unsupervised learning (such as clustering); therefore, not many approaches are available for feature selection for clustering in the case of hybrid and embedded categories. The authors in [20] provide an overview of feature selection methods for clustering. One of those methods, introduced by Dy and Brodley [17], was used in this paper.

The idea is as follows: sequential forward search is implemented, which starts from an empty feature set and adds features to the feature set one by one. For every feature, the expectation-maximization (EM) algorithm is run with initialization values obtained by the sub-sampling initialization algorithm. This algorithm takes 10% sub-sampling and has 10 sub-sampling iterations. In every iteration, k-means algorithm is run on sub-sample; if one of the clusters is empty, it is restarted, and the centroid of an empty cluster is set to the data point furthest from its cluster centroid. After obtaining centroids for every sub-sample, they are combined into one set, which is then clustered again using k-means, resulting in new centroids. The centroid (from the new set of centroids) that maximizes the likelihood of a combined set of centroids is selected as the initial cluster centers for the EM algorithm.

For every feature set in the sequential forward search, EM is run, and when it converges, the value of the criterion function is compared to the values of criterion functions for every feature set with the same number of features as the one currently observed. The feature with the best criterion function value is selected, and a new feature set is formed, which now includes the selected feature. Once again, all the other features are added one by one, and the procedure is repeated.

When one iteration of passage through every feature is completed and the feature set is obtained (e.g., with two features), the procedure is repeated and a feature set with one more feature is obtained (in this case, with three features). In order to compare the criterion values for feature sets of different dimensions, the cross-projection method of normalization is used. The sequential forward search is run as long as the criterion function is improved by the addition of another feature.

Another question that arises is what is the number of clusters that should be selected. In this procedure, the number of clusters is automatically selected. A maximum number of clusters is given, and then, for every feature set and for every K from the maximum number of clusters to 2, EM is run as described. Since the estimate of the maximum likelihood increases as more clusters are used, a Bayesian Information Criterion penalty term is added to the log-likelihood criterion.

4 Tree operators for random walk

In this section, new tree operators are proposed and their usage is explained and justified.

While solving a certain problem using GP, oftentimes, tree representation is used. Sometimes, the need arises to measure the distance between two individuals, and for that purpose, one can use Tree Edit Distance, which is calculated as the minimum cost sequence of node edit operations that transform one tree into another [41]. The problem with existing tree edit distances is that they do not take into account the fact that the nodes of the trees can have different degrees of importance. For example, in the case of symbolic regression, the tree individual has two types of nodes: function nodes and terminal nodes. So, one has to be careful while performing edit operation on the tree because if a function node is replaced by a terminal node, the resulting tree will not be feasible. Additionally, if there is a need to do a random walk through tree individuals of some search space, it is not clear how to obtain a feasible individual whose distance from the current individual is exactly one. That is why the new way of changing the tree individual has been designed such that it pays attention to the type of

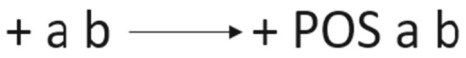

Fig. 1 Insert a function that has one child node

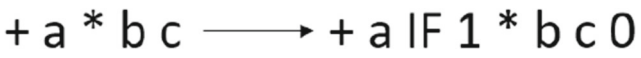

Fig. 3 Insert a function that has three child nodes

the node (function or terminal) and changes it accordingly. Three operators were designed for that purpose: insert, edit, and delete a node, and they have restrictions in order to produce feasible trees. The way in which those operators work is discussed below along with a description of the heuristic that measures the distance between any two trees regarding defined operators.

4.1 Description of operators

In this subsection, it is assumed that trees are denoted in prefix annotation. These operators were defined for our predefined subset of functions, but they can be expanded to different functions using the same rules if needed. The functions used here are shown in Table 1.

The **insert** operator allows the insertion of only a function node. That is because if a terminal node is added, the feasibility of the tree will be disrupted and there is no obvious way how to correct it and make it feasible again. The procedure is as follows: first, the node in front of which the new function node will be inserted is selected. If the new function node only needs a single child node, then the new node is only written down before the selected node. An example can be seen in Fig. 1 where POS is a function that only has one child. If the new function node needs to have two child nodes, the new node is written down before the selected node. Then, depending on the new node, a 0 or 1 is written down after all the children of the selected node. If the new function node is $+$ or $-$, then 0 is written down after the child nodes of the selected node because 0 is a neutral element for addition and subtraction. If the new function node is $*$ or $/$, then 1 is written down after the child nodes of the selected node because 1 is a neutral element for multiplication and division. The examples can be seen in Fig. 2.

If the node for insertion is IF function (the only one with three child nodes in the experiments conducted), then the selected node and all its children become the middle branch of IF, while 1 is put as a left and 0 as a right child of IF. The example can be seen in Fig. 3.

The last restriction for this insert operator is that it cannot insert function nodes as the leaves of a tree.

The **edit** operator exchanges the selected node with one of the allowed nodes. First, a node from the tree is selected. If the selected node is a function node, it can be replaced by another function node as long as those two function nodes have the same number of child nodes. If the selected node is a terminal node, it can be replaced by any other terminal node. The examples are given in Fig. 4a and b.

The **delete** operator can delete any tree node except the root node. During deletion, a node from the tree has to be selected at first. If the selected node is a terminal node, it will be deleted and in its place, 0 or 1 will be placed. If the parent of the deleted node is $+$, $-$, IF, or any of the function nodes that has only one child, then 0 will be put in the place of the deleted node. If the parent of the deleted node is $*$ or $/$, 1 will be placed instead of the deleted terminal. The example is given in Fig. 5a.

If the selected node is a function node, that node as well as all its children are removed from the tree. After that, in the place of selected node, 0 or 1 is placed in the same way as above. The example is given in Fig. 5b. If the selected node is a function node but of arity one, then only that node should be deleted without deleting any of its children nodes because such deletion will not violate the feasibility of a tree.

4.2 Description of distance heuristic

In practice, it is important to have ways of measuring the distance between any given trees. A heuristic that takes into account previously described operators is provided in this section.

For any two given trees in prefix annotation, the tree with more nodes is to be denoted as tree 1 and the other tree is to be denoted as tree 2 (if both trees have the same number of nodes, then one can choose which tree will be tree 1). Tree 1 and tree 2 are the input for the heuristics for which the pseudocode is given in Algorithm 1.

Fig. 2 Insert a function that has two child nodes. **a** Insert / node. **b** Insert – node

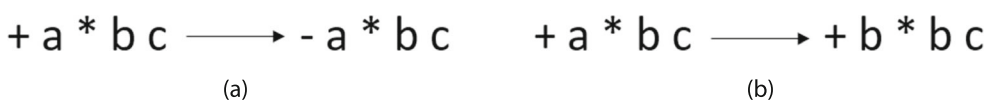

Fig. 4 Edit operator. **a** Edit function node (change + to –). **b** Edit terminal node (change a to b)

Algorithm 1 Pseudocode for the tree distance heuristic.

```
Input: tree1, tree2
Output: distance between tree1 and tree2
distance ← 0;
if tree1 == tree2 then
    return distance;
end
if tree1[0] ≠ tree2[0] then
    if arity of tree1[0] == arity of tree2[0] then
        EDIT tree1[0] to tree2[0];
    else
        INSERT tree2[0] as root of tree1;
    end
    distance ← 1;
end
for i ← 1 to length of tree1 do
    if tree1[i]≠tree2[i] then
        if tree1[i] ∈ terminals and tree2[i] ∈ terminals
        then
            EDIT tree1[i] to tree2[i];
        else if tree1[i] ∈ functions and tree2[i] ∈
        functions then
            if arity of tree1[i] == arity of tree2[i] then
                EDIT tree1[i] to tree2[i];
            else
                INSERT tree2[i] at i-th position of
                tree1;
            end
        else if tree1[i] ∈ terminals and tree2[i] ∈
        functions then
            INSERT tree2[i] at i-th position of tree1;
        else
            DELETE tree1[i];
            i ← i-1;
        end
        distance ← distance + 1;
    end
end
return distance;
```

An example of measuring distance between two arbitrary trees by using the given heuristic can be seen below. The functions that are used in the example are $+$, $/$, $*$, and POS, and the terminals are RS, D, SPD, TSC, and RR. In every step of the given example, a pair of nodes that is currently being compared is denoted as bolded and underlined.

T1: **/** + RS TSC + POS RS POS * D RR
T2: ***** POS / RS D + POS SPD RS
edit / to * → distance = 1

T1: * **+** RS TSC + POS RS POS * D RR
T2: * **POS** / RS D + POS SPD RS
insert POS before + → distance = 2

T1: * POS **+** RS TSC + POS RS POS * D RR
T2: * POS **/** RS D + POS SPD RS
edit + to / → distance = 3

T1: * POS / **RS** TSC + POS RS POS * D RR
T2: * POS / **RS** D + POS SPD RS
same terminal nodes → distance = 3

T1: * POS / RS **TSC** + POS RS POS * D RR
T2: * POS / RS **D** + POS SPD RS
edit TSC to D → distance = 4

T1: * POS / RS D **+** POS RS POS * D RR
T2: * POS / RS D **+** POS SPD RS
same function nodes → distance = 4

T1: * POS / RS D + **POS** RS POS * D RR
T2: * POS / RS D + **POS** SPD RS
same function nodes → distance = 4

T1: * POS / RS D + POS **RS** POS * D RR
T2: * POS / RS D + POS **SPD** RS
edit RS to SPD → distance = 5

T1: * POS / RS D + POS SPD **POS** * D RR
T2: * POS / RS D + POS SPD **RS**
delete POS → distance = 6

T1: * POS / RS D + POS SPD ***** D RR
T2: * POS / RS D + POS SPD **RS**
delete * → distance = 7

T1: * POS / RS D + POS SPD **0**
T2: * POS / RS D + POS SPD **RS**
edit 0 to RS → distance = 8

It can be seen that 8 steps are needed to reach from tree 1 to tree 2, so the distance between them is equal to 8.

Fig. 5 Delete operator. **a** Delete terminal node (a). **b** Delete function node (–)

5 Experiments

The main idea of this work was to group problem instances into similar clusters based on fitness landscape features and to, then, determine optimal parameters for GP on every cluster, not by hand but by using automatic algorithm configuration. This section explains how the clustering was made and how the GP parameters were obtained. Finally, the results will be shown.

5.1 Clustering using fitness landscape features

As the authors in [30] stated, while solving heterogeneous instances of a problem, it is perhaps better to group those instances with regard to a certain problem feature and to, then, try and find the best parameters for a similar group of instances than to try and find parameters that are the best for all problems at once.

In the case of the single machine environment, 100 train instances and 600 test instances were defined in a manner that is similar to [22]. For each scheduling instance, the number of jobs, their processing times, due dates, and weights were defined. Job durations may take integer values between 1 and 100, while their weights can take values between 0.01 and 1 in steps of 0.01. In order to generate the due dates for the jobs, two parameters were used: T (due date tightness), which represents the expected percentage of late jobs, and R (due date range), which represents the dispersion of due date values. Using those two parameters, the due dates were calculated using the following expression: $d_j \in [\sum_{j=1}^{n} p_j(1 - T - R/2), \sum_{j=1}^{n} p_j(1 - T + R/2)]$ where n represents the number of jobs in the test instance. For all train and test instances, the number of jobs in an instance takes the values of 12, 25, 50, or 100, while the parameters T and R take values of 0.2, 0.4, 0.6, 0.8, and 1 in various combinations.

For the unrelated machines environment, 60 train instances and 60 test instances were defined. As in the single machine environment, for each scheduling instance, the number of jobs, their processing times, due dates, and weights were defined along with the release dates for the jobs. Job durations, weights, and processing times were defined equally as in the case of the single machine environment. The difference being that processing times must be defined for every pair of job and machine; therefore, $m \times n$ processing times were defined where n is the number of jobs in an instance and m is the number of machines in an instance. The due dates were generated from the interval $[\hat{p}(1-T-R/2), \hat{p}(1-T+R/2)]$ where $\hat{p}$ is calculated as $\hat{p} = (\sum_{j=1}^{n} \sum_{i=1}^{m} p_{ij})/(m^2)$ and represents the estimated total processing time. The parameters T and R have the same meaning as in the case of the single machine environment. Additionally, the release times were chosen randomly from the interval $r_j \in [0, \frac{1}{2} \sum_{i=1}^{n} p_i]$. Here, for all train and test instances, the number of jobs in an instance takes the values of 12, 25, 50, or 100, while the parameters T and R take values of 0.2, 0.4, 0.6, 0.8, and 1 in various combinations. Additionally, the number of machines in instances were taken from the values 3, 6, 10, 15, or 20.

In RCPSP, the focus was on problems with 90 activities. Instances differ with regard to network complexity, resource factor, and resource strength; therefore, it seemed logical to try and find fitness landscape features that would help differentiate one particular problem instance from another. The test cases were obtained from the project scheduling problem library.[1] The test cases were divided into the train set (288 instances) and the test set (192 instances).

After fitness landscape features described in Section 3 of the given problems have been calculated, EM algorithm [36] with Gaussian mixture was used for clustering those problems. The following procedure was used: first, for every problem instance in the train set, 30 individuals were randomly initialized. From every individual, a random walk of length 1000 was calculated using the operators described in Section 4. Every tree in a random walk was evaluated, and the fitness values obtained in that evaluation, were used for calculating the fitness landscape features. In this way, 30 sets of every feature for each problem were obtained. The mean and the standard deviation were calculated; therefore, there were only two values for every feature in a problem. After obtaining those values, clustering was made for every fitness landscape feature using the mean and the standard deviation together (clustering based on two variables) as well as by using the mean and the standard deviation separately (clustering based on one variable). In this way, 22 clusterings based on two variables and 44 clusterings based on only one variable were obtained. For each of those clusterings, the problem that arised is how to determine the number of clusters. For solving that problem, 10 runs were made for each clustering, and in each run, the Bayesian Information Criterion (BIC) and the Akaike Information Criterion (AIC) were calculated for the number of components from 1 to 6. After all 10 runs were completed, the model with the lowest BIC value among all the runs and the model with the lowest AIC value among all the runs were selected. Between those two models, the one with fewer components was selected and used to fit the data and obtain clusters. Table 5 shows log likelihood and the number of clusters for features that had the best log likelihood values for the RCPSP. Column FL feature contains the fitness landscape features from the set of all calculated fitness landscape features that had the largest log likelihood values in EM clustering. For every feature in the table, the number of clusters with the lowest

[1] http://www.om-db.wi.tum.de/psplib/getdata_sm.html

Table 5 Cluster numbers and log likelihood based on mean and standard deviation for features: lengthscale median, lengthscale without zeros lower quartile, lengthscale average, lengthscale standard deviation, lengthscale without zeros standard deviation, lengthscale without zeros median, lengthscale without zeros average, lengthscale upper quartile, lengthscale without zeros upper quartile, partial information content and rate of neutral neighbors

FL feature	# of clusters	log likelihood
lengthscale median	4	9.891009425
lengthscale w/o 0s lq	3	9.519606422
lengthscale average	3	9.209791325
lengthscale stdev	3	8.853975226
lengthscale w/o 0s stdev	3	8.760618422
lengthscale w/o 0s median	3	8.732951897
lengthscale w/o 0s avg	3	8.688751344
lengthscale uq	4	8.424637449
lengthscale w/o 0s uq	3	8.17710097
partial information content	3	7.528115953
rate of neutral neighbours	3	7.254059803

AIC or BIC value is given in column # of clusters. The third column contains log likelihood, which was calculated by the EM algorithm for a given number of clusters for appropriate fitness landscape feature. The bigger the log likelihood, the more probable it is that the observed data fit Gaussian mixture obtained by the EM algorithm. It can be seen that almost every fitness landscape measure divides problem instances into 3 clusters, while only lengthscale median and lengthscale upper quartile divide instances into 4 clusters. It is interesting to notice that for the RCPSP, lengthscale measures give the best clusterings based on obtained log likelihood. Further experiments for the RCPSP were made for clustering obtained by *lengthscale median*, *lengthscale without zeros lower quartile* and *lengthscale average* because the obtained log likelihoods for those three fitness landscape measures are the greatest.

The use of the EM algorithm for the single machine environment and the unrelated machines environment did not yield good results, so instead, the k-means algorithm was used for clustering data in those two cases. Again, as in the case of the EM algorithm, the main problem was determining the number of clusters for the chosen fitness landscape features. For every fitness landscape feature (containing the average and standard deviation of values for that feature), the silhouette score [47], the Davies–Bouldin index [11], and the Calinski and Harabasz index [8] were calculated for the number of clusters from 2 to 10. The silhouette score for each sample was calculated as the difference between the mean intra-cluster distance and the mean nearest-cluster distance divided by the maximum value between those two distances. Its values vary from -1 to 1, and the bigger score means better clustering. Davies–Bouldin index is defined as the average similarity measure of each cluster with its most similar cluster where similarity is the ratio of within-cluster distances to between-cluster distances. In this case, lower values indicate better clustering. The Calinski and Harabasz index is defined as the ratio between the within-cluster dispersion and the between-cluster dispersion. As in the case of the silhouette score, larger values indicate better clustering. After the inspection of the obtained scores, *lengthscale average* with 3 clusters and *lengthscale without zeros maximum* with 3 clusters were selected for the single machine environment. In the case of the unrelated machines environment, *lengthscale without zeros standard deviation* with 6 clusters was selected.

5.2 Genetic programming parameters configuration

As mentioned previously, while using hyperheuristic such as GP, it is important to determine algorithm parameters in order to arrive at better solutions in the end. One can determine those parameters manually or use some method of automatic algorithm configuration. In order to get baseline algorithm parameters, they were determined manually on all instances from the train set (without clustering). The parameters that were tuned are the number of individuals in the population, mutation probability, maximum depth of a tree, and the size of the tournament in the tournament selection. For RCPSP, the number of generations was used as a stopping criterion, and it was set to 25 generations. In order to get the best parameters manually, every other parameter was fixed and the one that needs to be determined is varied. For every parameter that needs to be tuned, GP was run 30 times with every parameter value from the value set chosen in advance. The parameter value that yielded the smallest median of the criterion function values was chosen for further experiments. So, the number of individuals was chosen from set $\{250, 500, 750, 1000, 1024, 1500\}$, and it was determined that the best number of individuals is 1024. After that, the mutation probability was selected from set $\{0.1, 0.2, 0.3, 0.4, 0.5\}$, and it was determined that mutation probability should be 0.5. The maximum tree depth was chosen from set $\{3, 4, 5, 6, 7\}$, and it was determined that the maximum tree depth should be 5. At last, the tournament size was chosen to be 5 from the set $\{3, 4, 5, 6, 7, 8, 9\}$. Table 6 shows the parameters that were used for GP in experiments "by hand" for RCPSP.

For the single machine and the unrelated machines environment, the same procedure was carried out for determining GP parameters manually, and the obtained parameters are shown in Tables 7 and 8. In these two cases, the number of generations is set to 30. The stopping criterion was

Table 6 Parameters for genetic programming and RCPSP obtained manually

Parameter	Initial values	Optimized values
population size	1000	1024
mutation probability	0.3	0.5
maximum tree depth	7	5
tournament size	3	5

met when the algortihm reached that maximum number of generations.

The package that was used for automatic algorithm configuration was the *irace* package for R [32]. The *irace* package implements iterated racing procedure. It receives a parameter space definition, a target algorithm that will be run, and a set of instances on which it will tune the parameters. It then executes the target algorithm on different instances and with different parameter configurations; in this way, it carries out the exploration of the parameter search space in order to obtain good performing algorithm configurations. For every clustering and for every cluster, *irace* was run on the corresponding train set in order to obtain good GP parameters. As in the case of determining parameters manually, *irace* was tasked to determine the number of individuals in the population, the mutation probability, the maximum depth of a tree, and the size of the tournament in tournament selection. The number of individuals in the population was chosen from the integer set [200, 2000], the mutation probability was chosen from the set [0.01, 0.99], the maximum tree depth was chosen from the integer set [3, 10], and the tournament size was chosen from the integer set [3, 7]. The parameter configuration for *irace* is depicted in Table 9. The only difference in the parameter space was noted for the number of individuals in the population for the single machine and the unrelated machines environment. Unlike RCPSP where the maximum number of generations was set to 25, in these two cases, the maximum number of generations was set to 30; therefore, the interval from which the number of individuals in the population can be chosen was changed to [200, 1500].

Table 7 Parameters for genetic programming and the single machine environment obtained manually

Parameter	Initial values	Optimized values
population size	1000	1250
mutation probability	0.3	0.3
maximum tree depth	7	7
tournament size	3	3

Table 8 Parameters for genetic programming and the unrelated machines environment obtained manually

Parameter	Initial values	Optimized values
population size	1000	1000
mutation probability	0.3	0.3
maximum tree depth	7	5
tournament size	3	3

5.3 Results

After obtaining the parameters for every clustering and every cluster, GP was run 30 times on the corresponding train set both with manually determined parameters (i.e., the parameters that were determined manually on the entire train set before clustering) and parameters obtained by irace (for the corresponding cluster). The best individuals were then evaluated on the corresponding test set (instances from the initial test set that belonged to a cluster based on their fitness landscape features).

The results obtained for the three best clusterings of RCPSP (based on log likelihood as shown in Table 5; the larger the likelihood, the better the clustering) are shown below.

The parameters obtained by *irace* for every cluster for *lengthscale median* can be seen in Table 10.

It is observed that irace found different parameters for different clusters, which means that there is some difference between them, and that the fitness landscape feature *lengthscale median* managed to differentiate instances of a given problem. It is also interesting to note that the larger the population size, the smaller is maximum tree depth obtained. Table 11 shows the basic statistical measures (minimum, first quartile, median, mean, 3rd quartile, maximum, and standard deviation) of the fitness values for the individuals, which were obtained from 30 runs of GP using parameters obtained manually and parameters obtained by *irace* for clusters based on *lengthscale median*. Additionally, the table reports the p-value obtained by conducting the Wilcoxon sum rank test calculated in R. This test takes two samples and estimates the median of the difference between a sample from the first given vector and

Table 9 Parameter configuration for *irace*

Name	Switch	Type	Values
tsize	"–tsize"	i	(3, 7)
maxdepth	"–maxdepth"	i	(3, 10)
popsize	"–popsize"	i	(200, 2000)
mutProb	"–mutProb"	r	(0.01, 0.99)

Table 10 GP parameters obtained by *irace* for clusters based on *lengthscale median* feature for RCPSP

Parameter	Cluster 1	Cluster 2	Cluster 3	Cluster 4
Population size	733	1187	1320	1480
mutation probability	0.65	0.36	0.66	0.83
maximum tree depth	10	7	5	3
tournament size	3	3	7	5

Table 12 GP parameters obtained by *irace* for clusters based on *lengthscale without zeros lower quartile* feature for RCPSP

Parameter	Cluster 1	Cluster 2	Cluster 3
population size	884	1061	1911
mutation probability	0.35	0.28	0.4
maximum tree depth	8	6	7
tournament size	7	3	5

a sample from a second given vector. It is a non-parametric test that doesn't require data to be normally distributed. The null hypothesis is that the true location shift is equal to 0.

It can be seen that in the case of almost every cluster, the medians are smaller the for parameters from the *irace* run for the corresponding cluster, and the standard deviations are small in both cases. In cluster 1, the median is slightly smaller for manually obtained parameters, but the minimum value is better for the parameters obtained by using *irace*. Additionally, for every cluster, the minimum value is better for the parameters from *irace*. Boxplots can be seen in Fig. 6a-d.

The same behavior can be seen for the fitness landscape feature *lengthscale without zeros lower quartile*. Table 12 shows the parameters obtained by using *irace* for every cluster for *lengthscale without zeros lower quartile*. Again, the obtained values are different for different clusters. Here, the mutation probabilities are smaller than those in the clusters obtained for *lengthscale median*, which indicates that the clusters obtained by these two features indeed possess different characteristics.

Table 13 depicts the median of fitness values, the minimum fitness value, and the standard deviation of fitness values for individuals obtained by conducting 30 runs of GP using parameters obtained manually and by *irace* for clusters based on *lengthscale without zeros lower quartile*.

It can be observed that the median and the minimum value is the same or better for parameters obtained by *irace* on clustered data. The standard deviation of fitness values is approximately the same for parameters selected manually and those found by *irace* for every cluster. Instances in cluster 3 are those whose optimal value can be found; therefore, GP with parameters obtained manually and GP with parameters obtained by irace for a given cluster provide the same results. The boxplots for cluster 1 and cluster 2 can be seen in Fig. 7a-d.

Finally, Tables 14 and 15 depict the comparison between the parameter values and the fitness values for clusters obtained by using the fitness landscape feature *lengthscale average*, and Fig. 8a–d show the corresponding boxplots.

Table 14 shows that in cluster 3, *irace* resulted in a very small population size and relatively small tree depth. Although the population size is much smaller than the one selected manually, the median of the fitness values is smaller in the *irace* column of cluster 3 in Table 15 than in the column which contains results obtained with manually determined parameters, which means that better results are achieved in a smaller amount of time for that cluster. It can be seen that the medians are better or equal for all three clusters in the case of using parameters obtained by *irace*, but the minimum values are smaller for GP runs with manually obtained parameters in clusters 2 and 3. In cluster 1, obtained median values are the same in both cases which means that for instances in this cluster, GP with both sets of parameters managed to find best solutions. It can

Table 11 Comparison of fitness values for clusters based on *lengthscale median* for RCPSP

	Cluster 1		Cluster 2		Cluster 3		Cluster 4	
	manually	irace	manually	irace	manually	irace	manually	irace
min	1.69451	1.69409	2.56866	2.5623	1.84721	1.84313	2.5355	2.53248
1st qu.	1.695265	1.695938	2.594908	2.615698	1.85003	1.84794	2.553758	2.546463
median	1.696535	1.69694	2.658145	2.64807	1.85311	1.85005	2.559135	2.549805
mean	1.69688	1.697595	2.650749	2.651161	1.855899	1.852741	2.557996	2.552692
3rd qu.	1.69783	1.69858	2.687268	2.686103	1.860388	1.855968	2.56279	2.559048
max	1.70072	1.70725	2.75543	2.78116	1.88404	1.8768	2.57017	2.5817
stdev	0.001793	0.002618	0.054638	0.052508	0.008205	0.007859	0.007963	0.010326
p-value	0.8263		0.4646		0.03117		0.005608	

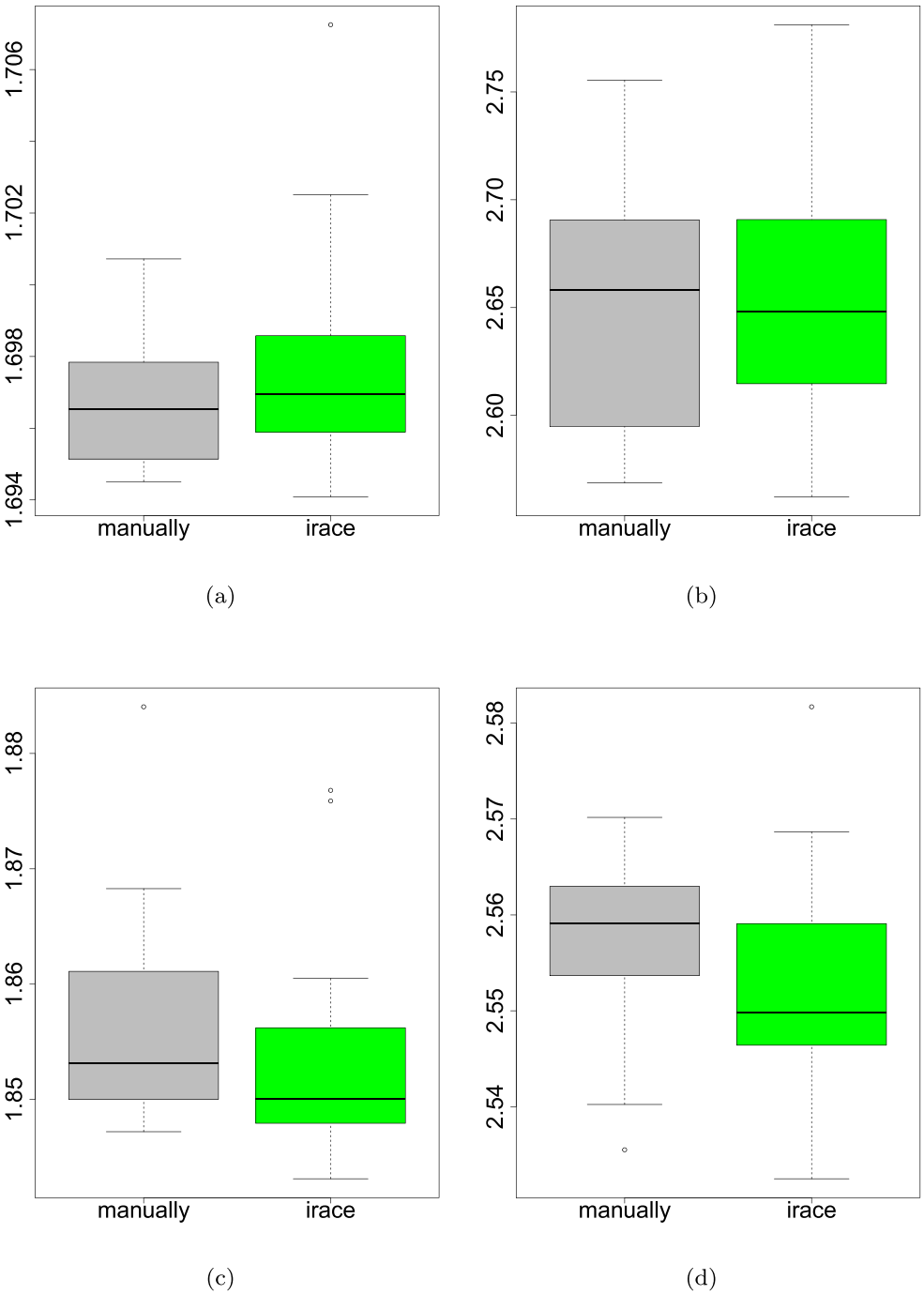

Fig. 6 Boxplots for clusters based on the average and the standard deviation of *lengthscale median* for RCPSP. **a** Cluster 1. **b** Cluster 2. **c** Cluster 3. **d** Cluster 4

be assumed that by getting better clusters (with greater log likelihood), one could obtain better results. For clustering using only one variable (mean or standard deviation of a fitness landscape feature), the results were not as good as in the case of clustering using two variables.

The results obtained by the same procedure as in RCPSP for clusters based on *lengthscale average* and *lengthscale without zeros maximum* for the single machine environment are presented below.

Table 16 shows the parameters obtained by *irace* for every cluster for *lengthscale average*. It can be observed that, here, irace also provided different parameters for different clusters. Although the values for population size, maximum tree depth, and tournament size are the same or very similar for clusters 1 and 2, the value of mutation probability differs significantly. Perhaps, this indicates that in cluster 2, there were fewer local optima, so the mutation probability did not need to be high in

Table 13 Comparison of fitness values for clusters based on *lengthscale without zeros lower quartile* for RCPSP

	Cluster 1		Cluster 2		Cluster 3	
	manually	irace	manually	irace	manually	irace
min	1.7936	1.7936	2.5024	2.4964	1.6555	1.6555
1st qu.	1.8493	1.8493	2.5249	2.5267	1.6555	1.6555
median	1.8878	1.8785	2.5311	2.5301	1.6555	1.6555
mean	1.8945	1.8813	2.5282	2.5292	1.6555	1.6555
3rd qu.	1.9172	1.9031	2.5343	2.5379	1.6555	1.6555
max	2.0093	2.0090	2.5438	2.5473	1.6555	1.6555
stdev	0.0508	0.0485	0.0112	0.0116	0.0000	0.0000
p-value	0.2041		0.5617		1	

order to promote extensive exploration over convergence to promising neighboring solutions.

Table 17 shows the basic statistical measures of fitness values for individuals obtained by conducting 30 runs of GP using parameters obtained manually and by *irace* for clusters based on *lengthscale average* in the single machine environment as well as the p-value obtained from the Wilcoxon sum rank test.

It can be observed that in all three obtained clusters, the fitness values obtained by running GP with the parameters from *irace* and the minimum, and median values are lower than those for the fitness values obtained by running GP with parameters determined manually. Additionally, the standard deviations are smaller in the *irace* columns of the table, which indicates that the results are less dispersed in the case where the parameters were determined automatically. It can be observed that in the case of all three clusters, mean fitness values that are achieved are much better for parameters obtained automatically than for manually obtained parameter values. This results in fewer outliers in fitness values obtained by the GP run with automatically determined parameters. Boxplots for the obtained clusters can be seen in Fig. 9a–c.

Table 18 depicts the parameters obtained by *irace* for every cluster for *lengthscale without zeros maximum*. It can be observed that in this case, irace also resulted in different parameters for different clusters and that in all three cases, the mutation value is high. In this case, *irace* concluded that there is a need to escape local minima, so the mutation probabilities are higher.

Table 19 shows the basic statistical measures of fitness values for individuals obtained by 30 runs of GP using parameters obtained manually and by *irace* for clusters based on *lengthscale without zeros maximum* in the single machine environment along with the p-value obtained by the Wilcoxon sum rank test.

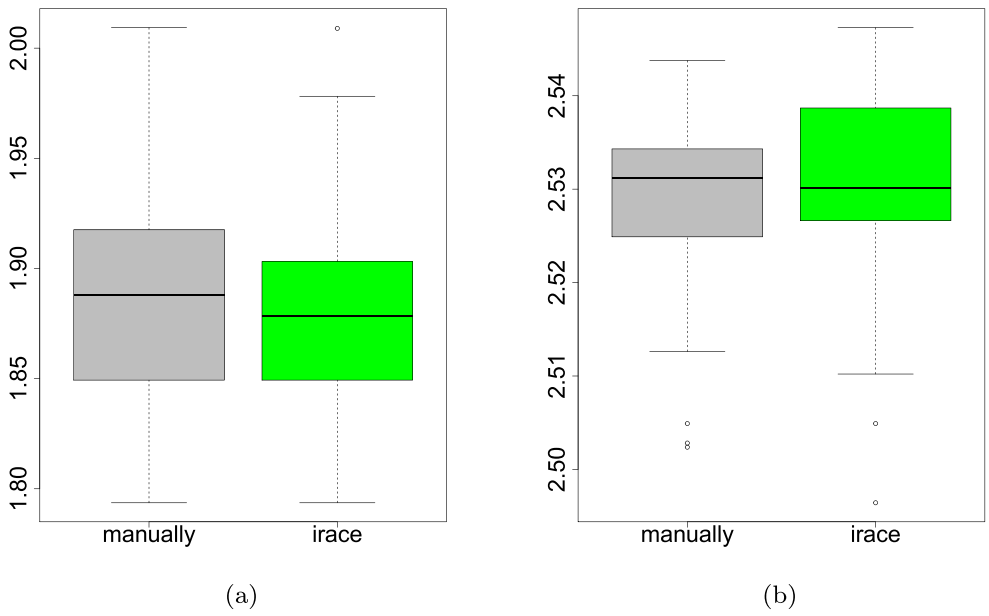

Fig. 7 Boxplots for clusters based on the average and the standard deviation of *lengthscale without zeros lower quartile* for RCPSP. **a** Cluster 1. **b** Cluster 2

Table 14 GP parameters obtained by *irace* for clusters based on *lengthscale average* feature for RCPSP

Parameter	Cluster 1	Cluster 2	Cluster 3
population size	1690	1844	278
mutation probability	0.89	0.79	0.18
maximum tree depth	3	5	3
tournament size	5	3	4

Table 15 Comparison of fitness values for clusters based on *lengthscale average* for RCPSP

	Cluster 1		Cluster 2		Cluster 3	
	manually	irace	manually	irace	manually	irace
min	1.65664	1.65664	1.7343	1.73701	2.50826	2.52411
1st qu.	1.65699	1.65699	1.738253	1.738943	2.563685	2.61086
median	1.65699	1.65699	1.74145	1.740645	2.6445	2.63489
mean	1.656981	1.657014	1.742834	1.743792	2.644536	2.658473
3rd qu.	1.656998	1.65699	1.745655	1.7455	2.71177	2.715253
max	1.65735	1.65766	1.75878	1.76969	2.79849	2.8256
stdev	0.000197	0.000223	0.006424	0.007657	0.087447	0.079728
p-value	0.4763		0.6049		0.6848	

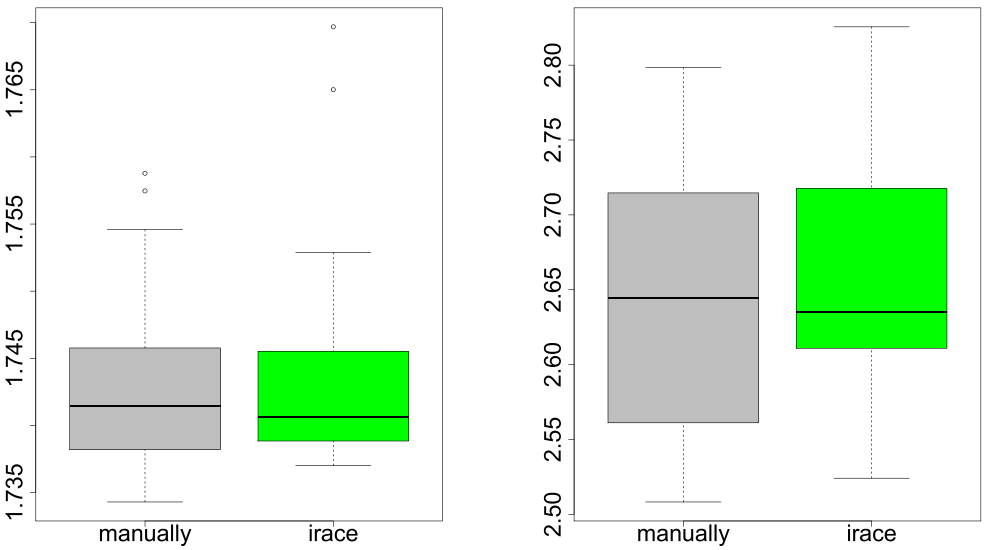

Fig. 8 Boxplots for clusters based on the average and the standard deviation of *lengthscale average* for RCPSP. **a** Cluster 2. **b** Cluster 3

Table 16 GP parameters obtained by *irace* for clusters based on *lengthscale average* feature for the single machine environment

Parameter	Cluster 1	Cluster 2	Cluster 3
population size	1492	1444	660
mutation probability	0.44	0.02	0.49
maximum tree depth	5	5	6
tournament size	7	7	5

Table 17 Comparison of fitness values for clusters based on *lengthscale average* for the single machine environment

	Cluster 1		Cluster 2		Cluster 3	
	manually	irace	manually	irace	manually	irace
min	5.7855	5.77658	1984.27	1980.23	68.0418	67.7454
1st qu.	5.919615	5.826545	1987.773	1987.288	68.97585	69.12405
median	5.954185	5.921155	1989.885	1988.64	70.029	69.92875
mean	5.976624	5.911362	1996.12	1990.591	69.94372	69.78849
3rd qu.	6.03374	5.946313	1996.61	1990.705	70.65295	70.51508
max	6.18379	6.09352	2039.49	2010.2	71.792	72.2765
stdev	0.101632	0.086486	15.44785	6.872063	1.142094	1.124739
p-value	0.006105		0.129		0.2413	

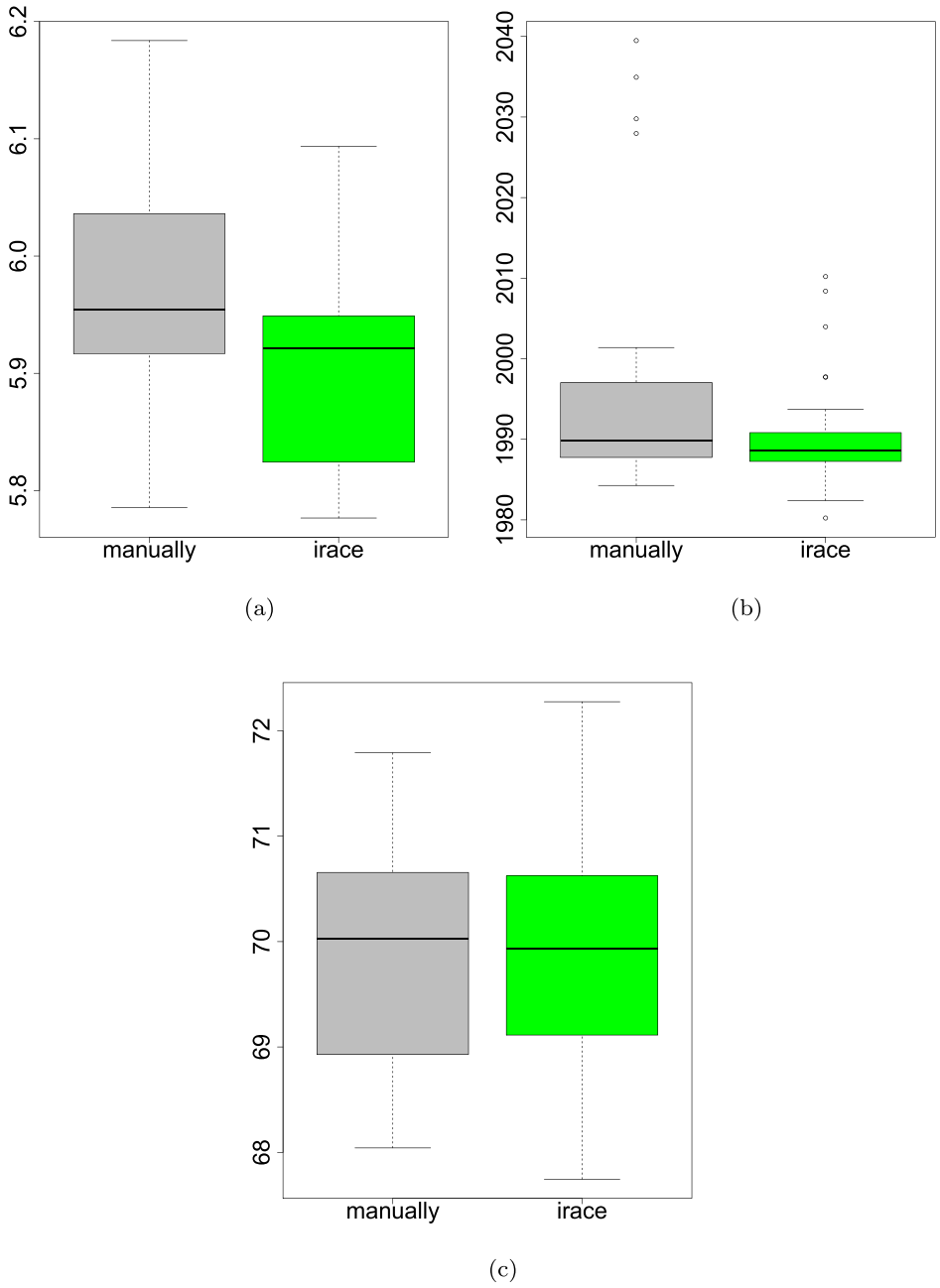

Fig. 9 Boxplots for clusters based on the average and the standard deviation of *lengthscale average* for the single machine environment. **a** Cluster 1. **b** Cluster 2.**c** Cluster 3

Table 18 GP parameters obtained by *irace* for clusters based on *lengthscale without zeros maximum* feature for the single machine environment

Parameter	Cluster 1	Cluster 2	Cluster 3
population size	850	955	1141
mutation probability	0.46	0.87	0.97
maximum tree depth	5	3	4
tournament size	6	8	7

It can be observed that in clusters 1 and 3, the medians of values are better for the parameters obtained by *irace*, while in cluster 2, the median of values is slightly better for the parameters obtained manually. However, in the case where *irace* provides better results, the improvement is 2.52% for cluster 1 and 0.38% for cluster 3, while the deterioration in cluster 2 is 0.099%. This indicates that the improvement is greater than deterioration, and therefore, these results exhibit a certain improvement of the fitness values, which is the goal of solving a problem by using hyperheuristics. Clusters 1 and 2 have much smaller dispersion of fitness values in the case of GP with parameters obtained by *irace* than in the case of GP with parameters obtained manually. It is also interesting to notice the large difference in medians of fitness values between clusters. The reason for such large diferences lies in the number of problem instances in test sets of obtained clusters. Cluster 1 has fewest instances, so the obtained values are the smallest, while cluster 2 has the most instances amongst the clusters, so the obtained fitness values are the largest. Boxplots for the obtained clusters can be seen in Fig. 10a-c.

Finally, the results obtained by using the same procedure as in the other two scheduling problems for clusters based on *lengthscale without zeros standard deviation* for the unrelated machine environment will be presented below.

Table 20 shows parameters obtained by *irace* for every cluster for *lengthscale without zeros standard deviation*. As in the case of the first two scheduling problems, *ivrace* produced different parameters for different clusters in this case as well. Here, the mutation probabilities vary from very low to very high values, which indicates that the obtained clusters have different landscape features since in some clusters, the mutation probability needs to be high in order to escape local minima, while in others, it can be kept low because there is less probability of being stuck in local minima.

Table 21 shows the basic statistical measures of fitness values for individuals obtained by 30 runs of GP using parameters obtained manually and by *irace* for clusters based on *lengthscale without zeros standard deviation* in the unrelated machine environment along with the p-value obtained by the Wilcoxon sum rank test.

It can be observed that in the case of four out of six clusters, the median values are smaller with parameters obtained by *irace* for a given cluster than with parameters obtained manually for all instances without clustering. In cluster 3, the median values are the same, while in cluster 4, the mean value in the *irace* column is slightly worse than that in the column where the results obtained by using manually obtained parameters are shown. Overall, the parameters obtained by *irace* achieve better results in most cases. Boxplots for the obtained clusters can be seen in Fig. 11a–f. In cluster 5, the y-axis is limited to values of up to 0.6; therefore, the outlier achieved in GP runs with manually obtained parameters is not shown.

It can be seen that in many of the problem instance clusters, there is no statistically significant difference between the two parameter configurations. There may be several reasons behind this kind of behaviour, which can contribute to the outcome. One of the possible causes is less than optimal grouping of problem instances in clusters due to inadequate landscape measures or their inability to separate problem instances into meaningful groups. This is certainly an area that should be explored in future work. The other reason may be the fact that parameter values do not play such an important role in certain clusters or problem instances. However, most of the time, automatically selected parameters yielded improved or at least a more stable performance, which

Table 19 Comparison of fitness values for clusters based on *lengthscale without zeros maximum* for the single machine environment

	Cluster 1		Cluster 2		Cluster 3	
	manually	irace	manually	irace	manually	irace
min	1.15895	1.18281	2020.43	2022.9	34.7672	34.7283
1st qu.	1.235658	1.22815	2024.183	2026.258	35.06853	35.12828
median	1.2878	1.25616	2028.205	2030.22	35.40675	35.27305
mean	1.294969	1.251622	2036.984	2037.446	35.44197	35.47329
3rd qu.	1.344593	1.284115	2034.938	2047.78	35.7262	35.77808
max	1.5156	1.34746	2229.93	2069.95	36.3611	37.153
stdev	0.079405	0.045844	37.41355	14.00665	0.468791	0.565669
p-value	0.01013		0.9614		0.4735	

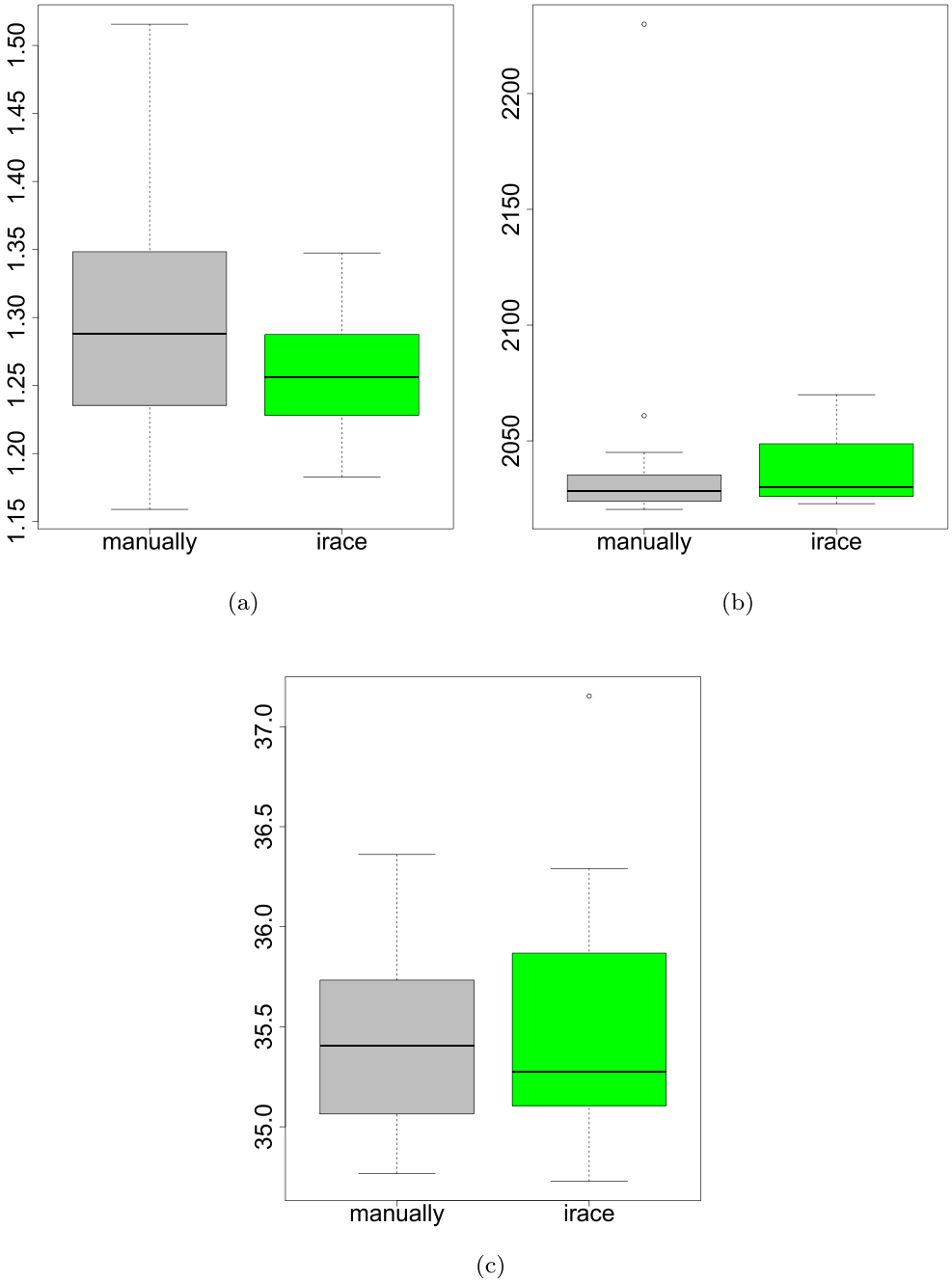

Fig. 10 Boxplots for clusters based on the average and the standard deviation of *lengthscale average* for the single machine environment. **a** Cluster 1. **b** Cluster 2. **c** Cluster 3

makes this a particularly useful property in regard to dealing with unseen problem instances. That is why we deem the use of the proposed approach while devising dispatching rules for solving scheduling problems using GP justified.

5.4 Time consumption

While presenting the fitness landscape features used in this paper, it was mentioned that one should use fitness landscape measures that can be calculated relatively fast. The main idea is that the process of calculating the measures should not take longer than the solving of the initial problem. So, a brief discussion about the time consumption of measures used for RCPSP in the experimental section is presented here. All time measurements are conducted on a machine with Intel i3-5005U CPU @ 2.00GHz processor and 8GB of RAM memory. On this machine, running genetic programming for unclustered data (i.e. on the entire train set) takes approximately 8384 seconds for 25 generations and 1024 individuals, which have been determined previously. On the other hand, calculating 30 random walks for one instance of a problem takes

Table 20 GP parameters obtained by *irace* for clusters based on *lengthscale without zeros standard deviation* feature for the unrelated machine environment

Parameter	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5	Cluster 6
population size	1165	1163	964	995	1165	610
mutation probability	0.42	0.3	0.11	0.46	0.03	0.86
maximum tree depth	7	5	8	5	7	8
tournament size	3	6	5	3	5	7

approximately 1.33 seconds, whereas calculating the distance between all pairs of trees in those random walks takes approximately 1819 seconds. To determine the fitness value of every tree in all the random walks, approximately 124.49 seconds is needed per instance. After calculating random walks and distances, one can calculate the lengthscale measures. To calculate the lengthscale median for one instance of a problem, it takes 50.8 seconds on average. Most of that running time is used on loading fitness values of trees in random walk and distances between those trees. It is observed that calculating lengthscale without zeros lower quartile takes 48.1 seconds on average, while calculating the lengthscale average takes 50.6 seconds on average. One should bear in mind that in this case, fitness landscape analysis has to be conducted only once; then, for every instance, one only has to determine the cluster that the instance belongs and use the parameters determined for that cluster to solve the given instance.

5.5 Feature selection

It is interesting to check whether some form of feature selection could select fitness landscape features to obtain clusters for which parameter tuning will be carried out.

For RCPSP data, the procedure described in Section 3.1 was run and the feature set with three features was obtained: *lengthscale without zeros minimum – average*, *lengthscale without zeros minimum – standard deviation* and *lengthscale median – standard deviation*. The number of obtained clusters was 8. After obtaining the clusters, the procedure described in Section 5.2 was carried out. Table 22 shows the parameters obtained by irace for every cluster.

Table 21 Comparison of fitness values for clusters based on *lengthscale without zeros standard deviation* for the unrelated machine environment

	Cluster 1		Cluster 2		Cluster 3	
	manually	irace	manually	irace	manually	irace
min	7.1251	7.15717	0.186766	0.1602	0	0
1st qu.	7.914348	8.012988	0.208729	0.213156	0	0
median	8.31334	8.20533	0.219857	0.217573	0	0
mean	8.25544	8.319772	0.24058	0.232507	0	0.001418
3rd qu.	8.596428	8.506123	0.248001	0.241289	0	0
max	9.43371	10.741	0.433637	0.458366	0	0.042551
stdev	0.579388	0.643819	0.057064	0.054607	0	0.007769
p-value	0.4383		0.3951		0.8493	
	Cluster 4		Cluster 5		Cluster 6	
	manually	irace	manually	irace	manually	irace
min	4.2349	4.31663	1.11114	1.12143	0.065203	0.065203
1st qu.	4.365478	4.500293	1.189258	1.199133	0.07583	0.071972
median	4.538645	4.575555	1.273115	1.24316	0.099468	0.078383
mean	4.525762	4.586342	1.284145	1.273132	0.711515	0.122222
3rd qu.	4.635878	4.642215	1.34004	1.314728	0.121269	0.114038
max	4.97366	5.14845	1.5318	1.63751	17.7828	0.575197
stdev	0.179976	0.16569	0.114342	0.117207	3.225743	0.102372
p-value	0.8915		0.305		0.1084	

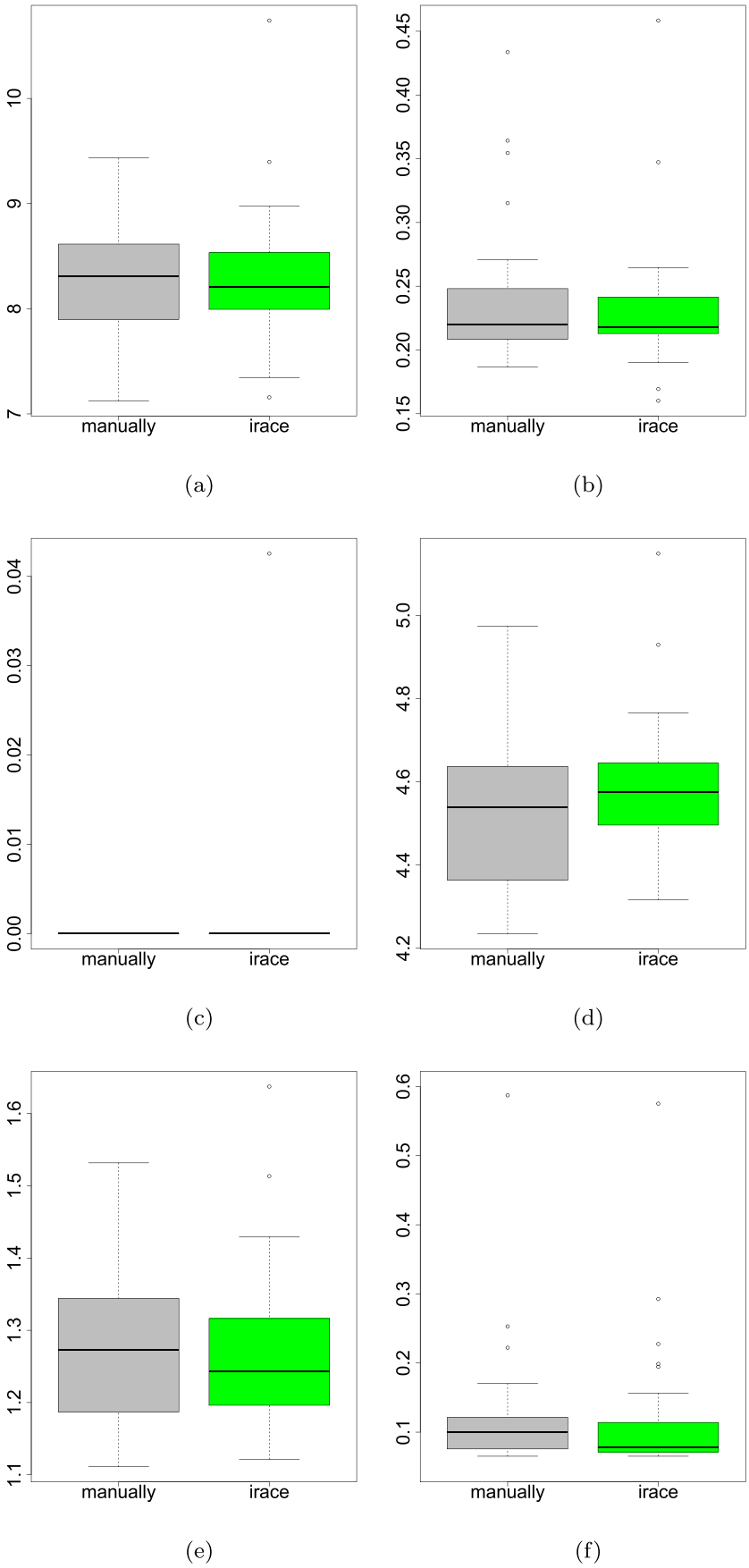

Fig. 11 Boxplots for clusters based on the average and the standard deviation of *lengthscale without zeros standard deviation* for the unrelated machine environment. **a** Cluster 1. **b** Cluster 2. **c** Cluster 3. **d** Cluster 4. **e** Cluster 5. **f** Cluster 6

Table 22 GP parameters obtained by *irace* for clusters based on feature selection

Parameter	cl. 1	cl. 2	cl. 3	cl. 4	cl. 5	cl. 6	cl. 7	cl. 8
population size	1039	505	861	809	1059	649	1219	840
mutation probability	0.95	0.16	0.44	0.31	0.34	0.63	0.78	0.47
maximum tree depth	6	3	5	7	6	8	5	9
tournament size	3	6	3	5	4	4	6	5

Again, it should be noted that the parameters differ for every cluster, which means that it makes sense to cluster the data into similar groups because for every cluster, a different set of parameters yields the best results. The comparison of the results obtained by running GP with parameters obtained manually and parameters obtained using irace can be seen in Table 23. The table shows the basic statistical features of the data and the p-value for the Wilcoxon sum rank test.

It can be observed that for 5 out of 8 clusters, the median is smaller in the case of results obtained by running GP using the parameters from *irace* experiments. Cluster 4 provides the same values for both cases. In the two clusters where the median is smaller with manually obtained parameters, the difference between medians is less than 0.0006, while in clusters where the median is smaller with parameters obtained by *irace*, the difference ranges from 0.002 to 0.0035. This shows that the improvement in those clusters is more pronounced than the deterioration in the other two clusters. Although the improvement is not achieved in every cluster and the p-values are not so small, it makes sense to use clustering and, then, parameter configuration based on fitness landscape features. The reason behind that is that an improvement can be achieved in this procedure, and that is the goal to be fulfilled while solving a problem using hyperheuristics.

Additional improvement could potentially be achieved by using some other form of feature selection or by using some other clustering algorithm; therefore, that is an interesting venue that can be explored in future research.

6 Conclusion

This paper shows one method of determining good parameters for solving scheduling problems using GP and fitness landscape analysis. It can be observed that the use of fitness

Table 23 Comparison of fitness values for clusters based on feature selection

	Cluster 1		Cluster 2		Cluster 3		Cluster 4	
	manually	irace	manually	irace	manually	irace	manually	irace
min	1.84558	1.84378	2.10707	2.09289	1.94486	1.94672	1.65552	1.65552
1st qu.	1.857833	1.856445	2.128553	2.133463	1.950133	1.950555	1.65552	1.65552
median	1.86268	1.861515	2.148625	2.14613	1.95592	1.9565	1.65552	1.65552
mean	1.862805	1.860745	2.155716	2.15353	1.956923	1.95795	1.65552	1.65552
3rd qu.	1.866308	1.86512	2.16709	2.16878	1.962218	1.966153	1.65552	1.65552
max	1.88494	1.88201	2.25051	2.24167	1.97834	1.97139	1.65552	1.65552
stdev	0.008688	0.007949	0.038177	0.034476	0.00884	0.007977	4.52E-16	4.52E-16
p-value	0.1996		0.5177		0.7471		1	
	Cluster 5		Cluster 6		Cluster 7		Cluster 8	
	manually	irace	manually	irace	manually	irace	manually	irace
min	1.77248	1.77248	1.74874	1.74877	2.37149	2.36339	2.25054	2.24787
1st qu.	1.778675	1.778615	1.750725	1.75061	2.37878	2.376708	2.267918	2.265555
median	1.7802	1.780715	1.757085	1.753745	2.3832	2.3826	2.278255	2.276605
mean	1.781644	1.782569	1.755553	1.754889	2.38476	2.382752	2.283204	2.28351
3rd qu.	1.78448	1.785683	1.75882	1.75882	2.38918	2.386295	2.289523	2.290583
max	1.79604	1.80388	1.76332	1.7691	2.41386	2.40907	2.36092	2.36092
stdev	0.005076	0.006516	0.004259	0.004941	0.010531	0.009923	0.024606	0.027312
p-value	0.5992		0.2017		0.2998		0.4267	

landscape features for clustering given problem instances leads to better results in GP runs. In order to calculate the fitness landscape features for this problem, three new operators are introduced, which enables us to obtain random walks for expression tree individuals. Additionally, a heuristic that can calculate the distance between any two given trees using those operators is proposed. The conducted experiments show that the better the clustering, i.e., the better the instances are divided into clusters, the better the results. This means that for future work, it would be useful to find good combinations of fitness landscape features that can lead to better clusterings or to try some different fitness landscape features that can be calculated relatively fast. Moreover, it may be promising to test this approach on other scheduling problems or any problem that uses GP and tree individuals in order to see if it will yield good results.

Funding This work has been partially supported by the Croatian Science Fundation under the project IP-2019-04-4333, which is titled Hyperheuristic Design of Dispatching Rules.

Availability of data and material The data for the resource constrained project scheduling problem (RCPSP) is a part of the PSPLIB library, which is opensource and available online. The data for single machine environment and unrelated machines environment is available on request.

Code availability Available on request.

Declarations

Conflicts of Interest/Competing Interests The authors declare that they have no conflict of interest.

References

1. Acharya D, Goel S, Asthana R, Bhardwaj A (2020) A novel fitness function in genetic programming to handle unbalanced emotion recognition data. Pattern Recogn Lett 133:272–279
2. Astarabadi SSM, Ebadzadeh MM (2019) Genetic programming performance prediction and its application for symbolic regression problems. Inf Sci 502:418–433
3. Artigues C, Demassey S, Neron E (2008) Resource-constrained project scheduling: Models, algorithms, extensions and applications. Wiley-ISTE
4. Bouziri H, Mellouli K, Talbi E-G (2011) The k-coloring fitness landscape. J Comb Optim 21(3):306–329
5. Bogon T, Poursanidis G, Lattner AD, Timm IJ (2013) Setting up particle swarm optimization by decision tree learning out of function features. In: Filipe J, Fred A (eds) Agents and Artificial Intelligence. Springer, Berlin, pp 72–85
6. Baykasoğlu A, Madenoğlu FS, Hamzadayı A (2020) Greedy randomized adaptive search for dynamic flexible job-shop scheduling. J Manuf Syst 56:425–451
7. Chan K, Kwong CK, Kremer GE (2020) Predicting customer satisfaction based on online reviews and hybrid ensemble genetic programming algorithms. Eng Appl Artif Intell 95:103902
8. Caliński T, Harabasz J (1974) A dendrite method for cluster analysis. Commun Stat 3(1):1–27
9. Campobello G, Dell'Aquila D, Russo M, Segreto A (2020) Neuro-genetic programming for multigenre classification of music content. Appl Soft Comput 94:106488
10. Cheng T, Zhong J (2020) An efficient memetic genetic programming framework for symbolic regression. Memetic Comput:1–17
11. Davies DL, Bouldin DW (1979) A cluster separation measure. IEEE Transactions on Pattern Analysis and Machine Intelligence PAMI-1(2):224–227
12. Dimopoulos C, Zalzala A (2001) Investigating the use of genetic programming for a classic one-machine scheduling problem. Adv Eng Softw 32:489–498
13. Djurasević M, Jakobović D, Knežević K (2016) Adaptive scheduling on unrelated machines with genetic programming. Appl Soft Comput 48:419–430
14. Djumić M, Šižejković D, Čorić R, Jakobović D (2018) Evolving priority rules for resource constrained project scheduling problem with genetic programming. Futur Gener Comput Syst 86:211–221
15. Đurasević M, Jakobović D (2020) Comparison of schedule generation schemes for designing dispatching rules with genetic programming in the unrelated machines environment. Appl Soft Comput 96:106637
16. dos Santos Coelho L, Bora TC, Klein CE (2014) A genetic programming approach based on lévy flight applied to nonlinear identification of a poppet valve. Appl Math Model 38(5):1729–1736
17. Dy J, Brodley C (2004) Feature selection for unsupervised learning. J Mach Learn Res 5:845–889
18. Giorgi M, Quarta M (2020) Hybrid multigene genetic programming - artificial neural networks approach for dynamic performance prediction of an aeroengine. Aerosp Sci Technol 103:105902
19. Gomes FM, Pereira FM, da Silva AF, Silva M (2019) Multiple response optimization: Analysis of genetic programming for symbolic regression and assessment of desirability functions. Knowl Based Syst 179:21–33
20. Hancer E, Xue B, Zhang M (2020) A survey on feature selection approaches for clustering. Artif Intell Rev:1–27
21. Hien NT, Tran CT, Nguyen XH, Kim S, Phai VD, Thuy NB, Manh NV (2020) Genetic programming for storm surge forecasting. Ocean Eng 215:107812
22. Jakobović D, Budin L (2006) Dynamic scheduling with genetic programming. Lect Notes Comput Sci 3905:73–84
23. Jiang X, Lee K, Pinedo ML (2020) Ideal schedules in parallel machine settings. Eur J Oper Res
24. Kauffman S (1992) The origins of order: Self-organization and selection in evolution. emergence.org, vol 15
25. Kevin Michell V, Kristjanpoller W (2020) Strongly-typed genetic programming and fuzzy inference system: An embedded approach to model and generate trading rules. Appl Soft Comput 90:106169
26. Kinnear KEJ (1994) Fitness landscapes and difficulty in genetic programming. In: Evolutionary Computation, IEEE World Congress on Computational Intelligence, pp 142–147
27. Klein R (2000) Bidirectional planning: improving priority rule-based heuristics for scheduling resource-constrained projects. Eur J Oper Res 127(3):619–638
28. Kolisch R (1996) Serial and parallel resource-constrained project scheduling methods revisited: Theory and computation. Eur J Oper Res 90(2):320–333
29. Koza JR (1992) Genetic programming: On the programming of computers by means of natural selection. MIT Press, Cambridge
30. Liefooghe A, Derbel B, Verel S, Aguirre H, Tanaka K (2017) Towards landscape-aware automatic algorithm configuration: Preliminary experiments on neutral and rugged landscapes. In: Hu B, López-Ibáñez M (eds) Evolutionary Computation in Combinatorial Optimization. Springer International Publishing, Cham, pp 215–232

31. Liu M-Y, Huai W-X, Yang Z-H, Zeng Y-H (2020) A genetic programming-based model for drag coefficient of emergent vegetation in open channel flows. Adv Water Resour 140:103582
32. López-Ibáñez M, Dubois-Lacoste J, Pérez Cáceres L, Stützle T, Birattari M (2016) The irace package: Iterated racing for automatic algorithm configuration. Oper Res Perspect 3:43–58
33. Lu S (1979) A tree-to-tree distance and its application to cluster analysis. IEEE Trans Pattern Anal Mach Intell PAMI-1(2):219–224
34. Mehr AD (2020) An ensemble genetic programming model for seasonal precipitation forecasting. SN Appl Sci 2:1821
35. Malan KM, Engelbrecht AP (2014) Fitness landscape analysis for metaheuristic performance prediction. In: Recent Advances in the Theory and Application of Fitness Landscapes. Springer, Berlin, pp 103–132
36. Moon TK (1996) The expectation-maximization algorithm. IEEE Signal Proc Mag 13(6):47–60
37. Morgan R, Gallagher M (2017) Analysing and characterising optimization problems using length scale. Soft Comput 21:1735–1752
38. Nguyen S, Zhang M, Johnston M, Tan KC (2019) Genetic programming for job shop scheduling. In: Evolutionary and Swarm Intelligence Algorithms. Springer International Publishing, Cham, pp 143–167
39. Ochoa G, Qu R, Burke EK (2009) Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO '09. Association for Computing Machinery, New York, pp 341–348
40. Ochoa G, Vazquez-Rodriguez JA, Petrovic S, Burke E (2009) Dispatching rules for production scheduling: A hyper-heuristic landscape analysis. In: 2009 IEEE Congress on Evolutionary Computation, pp 1873–1880
41. Pawlik M, Augsten N (December 2011) Rted: A robust algorithm for the tree edit distance. Proc VLDB Endow 5(4):334–345
42. Pinedo M (2012) Scheduling: Theory, algorithms, and systems. Springer, New York
43. Pitzer E, Affenzeller M (2012) A comprehensive survey on fitness landscape analysis. In: Recent Advances in Intelligent Engineering Systems. Springer, Berlin, pp 161–191
44. Pitzer E, Beham A, Affenzeller M (2013) Automatic algorithm selection for the quadratic assignment problem using fitness landscape analysis. In: Middendorf M, Blum C (eds) Evolutionary Computation in Combinatorial Optimization. Springer, Berlin, pp 109–120
45. Prugel-Bennett A, Tayarani-Najaran M (2012) Maximum satisfiability: Anatomy of the fitness landscape for a hard combinatorial optimization problem. IEEE Trans Evol Comput 16(3):319–338. https://doi.org/10.1109/TEVC.2011.2163638
46. Ragalo AW, Pillay N (2018) Evolving dynamic fitness measures for genetic programming. Expert Syst Appl 109:162–187
47. Rousseeuw PJ (1987) Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. J Comput Appl Math 20:53–65
48. Salgotra R, Gandomi M, Gandomi AH (2020) Time series analysis and forecast of the covid-19 pandemic in india using genetic programming. Chaos Solitons Fractals 138:109945
49. Tayarani-N. M, Prugel-Bennett A (2014) On the landscape of combinatorial optimization problems. IEEE Trans Evol Comput 18(3):420–434. https://doi.org/10.1109/TEVC.2013.2281502
50. Uy N, Chu TH (2020) Semantic approximation for reducing code bloat in genetic programming. Swarm Evol Comput 58:100729
51. Ventura S, Luna JM (2016) Pattern mining with evolutionary algorithms. Springer International Publishing
52. Vassilev VK, Fogarty TC, Miller JF (2000) Information characteristics and the structure of landscapes. Evol Comput 8:31–60
53. Weinberger E (1990) Correlated and uncorrelated fitness landscapes and how to tell the difference. Biol Cybern 63(5):325–336
54. Wright S (1932) The Roles of Mutation, Inbreeding, Crossbreeding and Selection in Evolution. Proceedings of the Sixth International Congress on Genetics, pp 365–366
55. Zou J, Yuan J (2020) Single-machine scheduling with maintenance activities and rejection. Discret Optim 38:100609
56. Zupančić J, Filipić B, Gams M (2020) Genetic-programming-based multi-objective optimization of strategies for home energy-management systems. Energy 203:117769. https://doi.org/10.1016/j.energy.2020.117769. https://www.sciencedirect.com/science/article/pii/S0360544220308768

Rebeka Čorić is a Ph.D. student at Faculty of Electrical Engineering and Computing in Zagreb. She received masters degree in Financial and business mathematics in 2014 at University of Osijek - Department of Mathematics, where she currently works as teaching assistant.

Mateja Đumić is a postdoc at the University of Osijek - Department of Mathematics. She received Ph.D. in 2020 at the Faculty of Electrical Engineering and Computing in Zagreb on the subject of designing priority rules for RCPSP with genetic programming.

Domagoj Jakobović is Full professor at Faculty of Electrical Engineering and Computing, University of Zagreb. He received B.S. degree in December 1996. and MS degree in December 2001. in Electrical Engineering. Since April 1997. he is a member of the research and teaching staff at the Department of Electronics, Microelectronics, Computer and Intelligent Systems of Faculty of Electrical Engineering and Computing, University of Zagreb. He received Ph.D. degree in December 2005. on the subject of generating scheduling heuristics with genetic programming.