Appl Intell (2016) 45:777–792
DOI 10.1007/s10489-016-0791-1

Speeding up global optimization with the help of intelligent supervisors

Grzegorz Pawiński[1] **· Krzysztof Sapiecha**[1]

Published online: 29 April 2016

Abstract It is shown in the paper that Developmental Genetic Programming is an efficient tool for evolutionary development of intelligent supervisors that solve an extension of Resource-Constrained Project Scheduling Problem. The extension assumes that resources are only partially available. It also assumes that renewable resources affect the project cost. The cost should be as low as possible and a deadline of the project must be met. This is apparent with regard to software houses and building enterprises. Computational experiments showed that supervisors find solutions of the problem much faster than other genetic approaches. A specific property of the supervisor is that it has various strategies of allocating the resources to the tasks. The supervisor uses the strategies in order to develop a procedure for producing the best schedule for the whole project. The analysis of the evolutionary process was performed and experimental results were compared with the optimal ones obtained by means of the exhaustive search method.

Keywords Project scheduling · Resource allocation · Global optimization · Evolutionary computations · Developmental genetic programming

✉ Grzegorz Pawiński
g.pawinski@tu.kielce.pl

Krzysztof Sapiecha
krzysztof.sapiecha@gmail.com

1 Department of Computer Science, Kielce University of Technology, al.1000-lecia P.P. 7, 25-314 Kielce, Poland

1 Introduction

The researchers have been focusing their attention on making the best use of scarce resources available since PERT (Program Evaluation and Review Technique) and CPM (Critical Path Method) were developed in the late 1950s [14]. Resource-constrained project scheduling problem (RCPSP) [18] addresses a task of allocating limited resources over time, in order to perform a set of activities subjected to constraints in the order, in which the activities may be executed. In the RCPSP resources are assumed to be constantly available during the project execution. An extension of the problem, where resources are only partially available since they can be involved in many projects, was investigated in [32, 33]. In [33, 34] it is shown that Developmental Genetic Programming (DGP) may be adapted for the evolutionary development of intelligent supervisors that solve the extension. The supervisors have their own strategies and aim at allocating resources to project activities. They take into consideration the availability of the resources to minimize the total cost of the project and complete it before a deadline (deadline problem [38]). The supervisor develops a procedure that allocates the resources to the activities. The best supervisor develops the best procedure.

The DGP and hyper-heuristics are relatively new approaches to optimization that aim at managing solution methods rather than solutions themselves. The DGP makes it possible to investigate all areas of the search space without being constrained by demands on the validity of the solution space [40]. Hyper-heuristics work at a higher level when compared with typical applications of meta-heuristics to the optimization problems. In other words, a hyper-heuristic can be understood as a (meta-) heuristic operating on lower level (meta-) heuristics. The hyper-heuristic has no knowledge of either solution domain or a function of the low-level heuristics and has an access only to the low-level heuristics domain [4, 22].

The next section of the paper contains a brief overview of related work. In Section 3 our research motivation is given. Section 4 briefly outlines the main idea of creating supervisors, and Section 5 focuses on the implementation of our method. Computational experiments conducted in order to evaluate our approach and compare it with other methods, as well as with the optimal results, are given in Section 6. The paper ends with conclusions.

2 Related work

Both exact and heuristic methods have been used to solve the RCPSP. The solutions based on dynamic programming e.g. [15], enumeration algorithm of [31], or branch and bound algorithms of [6, 7, 11] were the first to be used. However, heuristics have been preferred due to substantial limitations of the exact methods. According to [1], the optimal solution can be achieved by exact procedures only for small projects that contain fewer than 60 activities and are not highly constrained. Moreover, exact methods may require a significant amount of computation time. Therefore, heuristic approaches to the implementation of resource allocation optimization algorithms are desired to enhance the process. In-depth study of the performance of recent RCPSP heuristics can be found in [20]. Genetic algorithms (GAs) are some of the most commonly used meta-heuristics to generate a solution to the problem [1, 5]. However, out of five evolutionary based algorithms, GAs shows the worse performance for the RCPSP [10]. Authors in [13] propose a genetic programming (GP) as an option that makes it possible to find an acceptable solution to the problem within a reasonable time. The method has achieved good results for the generation of efficient programs in different domains, e.g. mathematical calculations, robot control, text recognition, etc [26]. Recently, other meta-heuristics including particle swarm optimization [41] or ant colony optimization [27, 29] have also been suggested. Most genetic approaches do not distinguish between a genotype, i.e. a point in a search space, and its phenotype, i.e. a point in a solution space [16, 17]. Thus, to solve many problems the restrictions on the creation of the genotype structure are imposed. A candidate solution is not only judged according to its fitness or quality but has to obey certain restrictions which exclude entire regions of solution space as infeasible. The exclusion of regions eliminates sequences of genes, which may lead to high-quality solutions. The GP approach established by Koza starts out from a tree representation of the program. Each language obeys certain grammatical rules. Thus constraints have to be followed by the expressions in the respective language. Constraints imposed on the solution space may lead to local hills or valleys which are difficult to overcome with traditional methods of optimization [3]. Searching in such a space may be time consuming and does not give any satisfactory results. Moreover, it is worth noticing that the evolution of a genotype that does not fulfill constraints may lead to a genotype that constructs a correct phenotype. There are several ways to deal with constraints in GAs [12, 30]. One of them uses genotype-phenotype mapping (GPM). It consists in transferring the search space, where unrestricted search operators can be applied, into the solution space, where the feasibility of solutions is guaranteed. Each genotype is allowed in search space, whereas an appropriate GPM provides for feasible solutions in a solution space only [3]. The mapping is critical to the performance of the search process [17]. In Koza's work only those search operators are used that allow for the production of valid program trees. In our approach, any search operator may be used.

The DGP has already been successfully applied in the design of electronic circuits, control algorithms [23], strategy algorithms in computer games and synthesis of embedded systems [9], etc. Many human-competitive results produced using runs of genetic programming with a developmental process are described in [25]. An idea of intelligent supervisors capable of solving a specific case of the RCPSP which consists in efficient rescheduling of the project tasks by using limited renewable resources was introduced in [37]. The self-adaptive supervisor that can maintain the optimality of the system in spite of disruptions that are likely to occur dynamically is developed in [36]. Applications of the DGP in optimization problems of real-time computer-based systems including scheduling of real-time tasks in multiprocessor systems, hardware/software co-design of distributed embedded systems and budget-aware real-time cloud computing may be found in [8].

3 Motivation

Mhring [28] states that the RCPSP is one of the hardest problems of Operational Research. Assuming initial resource workload, the problem better fits real-life project management problems. It is common in IT business to deal with more than one problem, for example, where managers have to use a resource-sharing approach. Developmental Genetic Programming [17, 40] is an adaptation of the GP [24] to optimization problems. In the DGP a genotype space (search space) and a phenotype space (solution space) are operated separately and a mapping (encoding) is used to derive a phenotype from a genotype. This separation enables the DGP to investigate all areas of the genotype space. The investigation is not constrained by the restrictions on the validity of the phenotype space [40]. In [33] an adaptation of the DGP to the evolution of the best supervisor, which creates a procedure for schedule development, instead of the schedule itself, is given. Genes contained in a genotype of the supervisor decide on consecutive steps of the procedure. The comparison of DGP with other genetic approaches shows that it quickly produces good quality results. For the projects containing 30 tasks, which is a relatively small number for practical applications, the computation time and the quality of final results of the adaptation and those of the GA were comparable. However, it was observed that the convergence of the adaptation was much faster. To achieve a good quality result it did not require as many generations as the GA and the GP. The results of the adaptation were better after 5 generations than the ones obtained by the GA after 100 generations. Moreover, uncorrected sample standard deviation of the population was lower than in other methods. This means that the adaptation does not require as many test reruns, because each individual is close to the best one. Thus, on average the method is better.

The RCPSP is frequently present in a high scale project management of software development, power plant building, and military industry projects that include design, development, and building of nuclear submarines [35]. Such projects are very complex, and may comprise many more tasks than 30. Therefore, the usability of the DGP adaptation to speed up global optimization of such projects should be investigated. The PSPLIB contains projects including only up to 120 tasks. However, it may be used to evaluate the strengths of the supervisors in case of a high scale project management.

4 Development of supervisors

A specific property of the supervisor is that it has strategies of allocating the resources to the tasks. These are, for example, use the fastest resource or use the cheapest resource. The supervisor uses the strategies to develop a procedure for producing the best schedule for the whole project [33, 37].

A genotype of the supervisor defines how the supervisor uses resource allocation strategies to create a project schedule. A genotype in the form of a tree comprises nodes representing genes that specify construction functions for a solution of the problem and edges that indicate an order in which these functions are executed (a procedure of construction]).[1] The GPM is performed by an execution of the procedure, starting from a root of the tree. So, if a target solution (a phenotype) is a sequence of tasks with allocated resources, the construction of the solution is a method for selecting a resource for the allocation of each task. The method depends on a genotype of the supervisor. The supervisor having the best genotype (with the optimal allocation) is developed in an evolutional manner. Details of the development of the supervisors are described in [34].

The cost of the project schedule reflects its fitness (a case of the deadline problem). Actually, the cost is a measure of the quality of the supervisor. The lower the cost the higher the quality of the supervisor. The project cost (C) is defined as follows:

$$C = T_p \cdot C_p + \sum_{j=1}^{r} \left(C_u(j) \cdot u_j + C_e(j) \cdot \sum_{i=1}^{m} T(i,j) \cdot e_{ij} \right) \quad (1)$$

where

$$u_j = \begin{cases} 0 & \text{if resource } j \text{ has no assigned tasks} \\ 1 & \text{if resource } j \text{ has at least one assigned task} \end{cases}$$

$$e_{ij} = \begin{cases} 0 & \text{if task } i \text{ is not assigned to resource } j \\ 1 & \text{if task } i \text{ is assigned to resource } j \end{cases}$$

T_p –	the project duration,
C_p –	the project operating cost,
r –	the number of resources in the library
$C_u(j)$ –	resource j unit cost,
$C_e(j)$ –	the cost of task execution by the resource j,
$T(i,j)$ –	the time of task i executed by the resource j,
m –	the number of tasks,

The indirect costs result from the employment of renewable resources. For example, an enterprise periodically pays salaries. The indirect costs are defined as the product of a resource employment cost per unit and occupation time of renewable resources (the first term). The indirect costs are constant during execution time of the whole project no matter how high a workload of a team developing the project is. The direct costs are associated with the consumption of non-renewable resources. These are materials or machines and tools that wear out during the execution of the project. Direct costs are present only when a resource is assigned to the project (the second term). They comprise a resource deployment cost together with a sum of execution costs related to all allocated activities.

5 Implementation

Our method[2] was implemented in Java. A description of methods used in the implementation process, and the correspondence between the development of the genotypes

[1] A genotype in classical GAs represents a solution of the problem. In the DGP a genotype comprises a procedure that constructs the solution.

[2] It will be called supervisor (SV) method later on

Table 1 Description of methods used in the implementation

Genotypes as vectors	Genotypes as trees
Create genotype: assign a randomly drawn number of the resource to each gene.	Create genotype: create a full-tree and assign a randomly drawn strategy and location of the division to each node.
Evaluate genotype quality: take the task graph and the genotype with resource numbers and create a project schedule, then save the results.	Evaluate genotype quality: take the tree-based genotype and cut off branches to fit to H, and repair invalid nodes. Afterwards, traverse the tree to create a list of strategies. Take the task graph and the list to create a project schedule, then, save the results.
Create a project schedule: find a list of ready-to-start tasks. Go through the following steps while the list is not empty: · take a task from the list and a corresponding resource number from the vector of resources, · remove the task from the list, · calculate the earliest start time based on its predecessors, · assign the task to the resource, · recalculate the list of ready-to-start tasks Finally, calculate the project duration and cost.	Create a project schedule: find a list of ready-to-start tasks. Go through the following steps while the list is not empty: · take a task from the list and a corresponding strategy from the list of strategies, · execute the strategy to calculate a resource to allocate, · remove the task from the list, · calculate the earliest start time based on its predecessors, · assign the task to the resource, · recalculate the list of ready-to-start tasks Finally, calculate the project duration and cost.
Assign a task to the resource: check tasks already assigned to the resource and insert the task as soon as possible, taking into account all precedence relationships and tasks from the initial schedule, which cannot be moved.	
Crossover: take two randomly drawn genotypes and a random cut position. Replace parts of the vector that were cut off and evaluate the quality of offsprings. Add those children who fulfill the time constraints to a new population.	Crossover: take two randomly drawn genotypes and a random cut positions. Replace subtrees that were cut off and evaluate the quality of offsprings. Add those children who fulfill the time constraints to a new population.
Mutation: take a randomly drawn genotype and a random gene from the vector. Choose a resource number at random and replace	Mutation: take a randomly drawn genotype and a random mutation mode. Basing on the mode do as follows:

Table 1 (continued)

Genotypes as vectors	Genotypes as trees
the gene. Evaluate the quality of the individual and add it to a new population when it fulfills the time constraints.	· set the strategy and the location of the division to random values in a randomly drawn tree node, · remove a random subtree, · add leaves at random position. Then, evaluate the quality of the individual and add it to a new population when it fulfills the time constraints.
Check the time constraints (after genotype quality has been evaluated): An individual fulfills the time constraints if the project is completed before the deadline. When this holds true the individual is accepted. However, the individual is also accepted when the deadline is missed but it has better quality (a lower cost) then its parent. Otherwise, the individual is rejected.	

in the form of trees, and in the form of vectors are given in Table 1.

6 Experimental results

In order to highlight its strong and weak points the SV method was tested on projects from PSPLIB [21]. However, in our experiments, only activities and their precedence relationships were taken from project instances contained in the PSPLIB. The renewable resources were randomly generated. As a result the resource deployment cost $C_u(j)$ and the execution cost of the activity $C_e(j)$ could differ up to 10 % from default values, which were 20 and 1, respectively. As they were general-purpose resources, they could execute each activity. The values of resource parameters were set as shown in Table 2. We tested all 480 projects. Each of them comprised 60 tasks. In each test instance, the resources already had their own randomly allocated schedule with 30 tasks (so-called initial schedule). Activities from an initial schedule could not be moved, and therefore, the resources were available only in some time periods. For each project instance, 10 independent runs were performed and the results were averaged. The number of generations was limited to 100 because the experimental results showed that the convergence of the SV method is fast enough (as presented in Section 6.3 for all problems considered in this paper).

Table 2 Values of resource parameters in the experiments

j	$C_e(j)$	$C_u(j)$
1	0,95	20
2	0,96	20
3	1,07	20
4	0,9	22
5	1,03	19
6	0,97	20
7	1,0	22
8	0,99	19
9	0,91	20
10	1,1	22

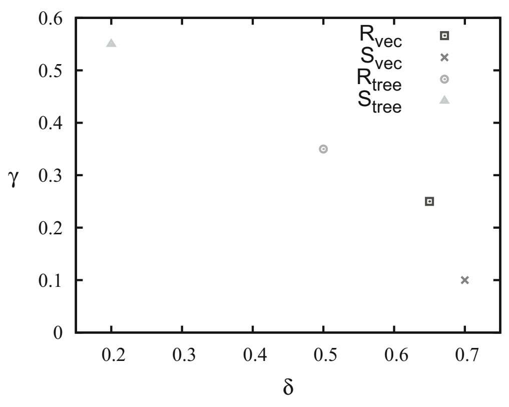


Fig. 1 Values of probabilities for the methods that gave the best results and were chosen for further tests

Table 3 The relationship between a height of the tree and the number of leaves

H	H_{max}	H_{avg}	Ln	Ln_{avg}	Un
6	6	4,85	32	6,41	350.657
8	8	5,48	128	7,73	594.680
10	10	5,83	512	8,57	747.538
12	10	6,03	2048	9,1	827.830
14	10	6,15	8192	9,43	865.492

H – the height of the initial tree, H_{max} –the maximum tree height, H_{avg} – the average tree height, Ln the number of leaves in the initial tree, Ln_{avg} - the average number of leaves, Un – the number of unique chromosomes

Evolution can be applied to a project schedule (like in GAs), to an algorithm to develop the schedule (like in the GP), or to a supervisor (schedule designer) that can optimally develop the algorithm (like in our adaptation of the DGP). In the first case one operates on a fixed-length string of chromosomes that use vectors of real numbers as individuals [39], i.e. one operates directly on resource numbers or decision strategies (like priority rules of [2, 19]). The second case evolves tree-shaped individuals strictly related to a solution space [39]. Binary trees that comprise a resource number in each node can represent tree-shaped individuals. The SV method fits into the third case, which uses the genotype-tree structures containing decision strategies and the genotype-phenotype mapping. However, we have not compared the results of the experiments with the optimal ones, although the PSPLIB was applied, because it was beyond the scope of our research. Therefore, in the experiments, we compared the use of tree-based methods vs. the use of a simpler representation of genotypes as vectors, and the use of decision strategies vs. the use of static assignment of resources to tasks, which does not require any additional computations. The components of our comparison were as follows:

– trees containing decision strategies (S_{tree}),
– trees containing resource numbers (R_{tree}),
– vectors of decision strategies (S_{vec}),
– vectors of resources (R_{vec}).

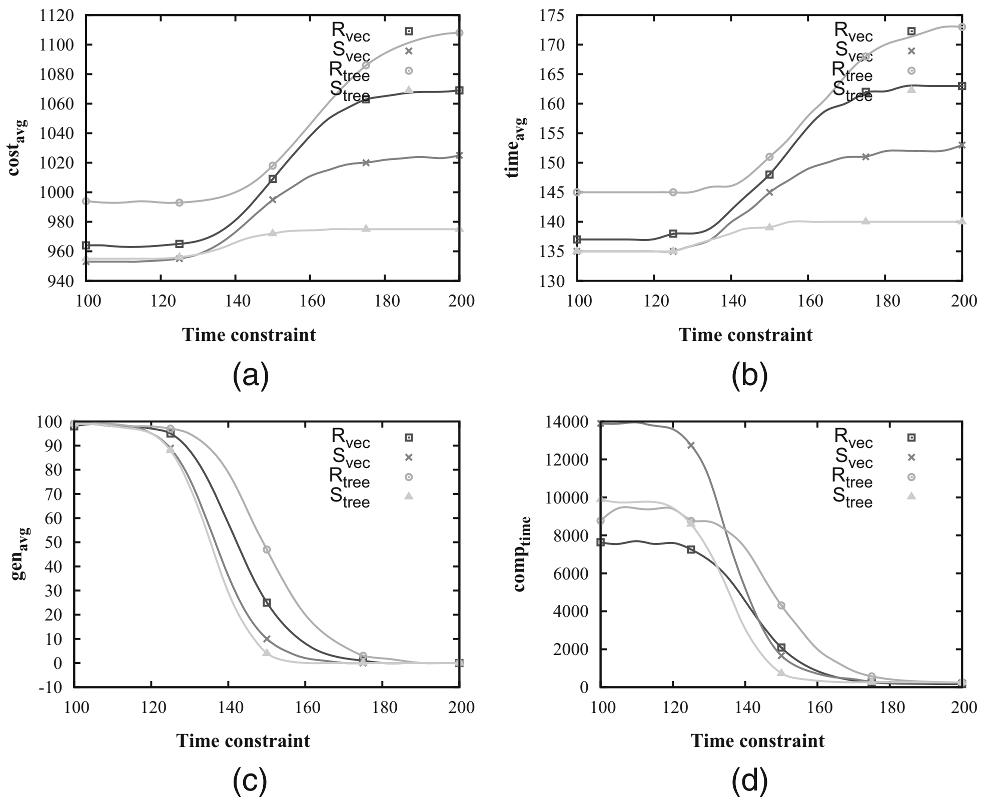


Fig. 2 Experimental results for different methods for $\Pi = 240$. $cost_{avg}$ – the average project cost; $time_{avg}$ – the average project duration from the best individuals in each test run, gen_{avg} – the average number of generations at which evolution stopped, $Comp_{time}$ – the average computation time

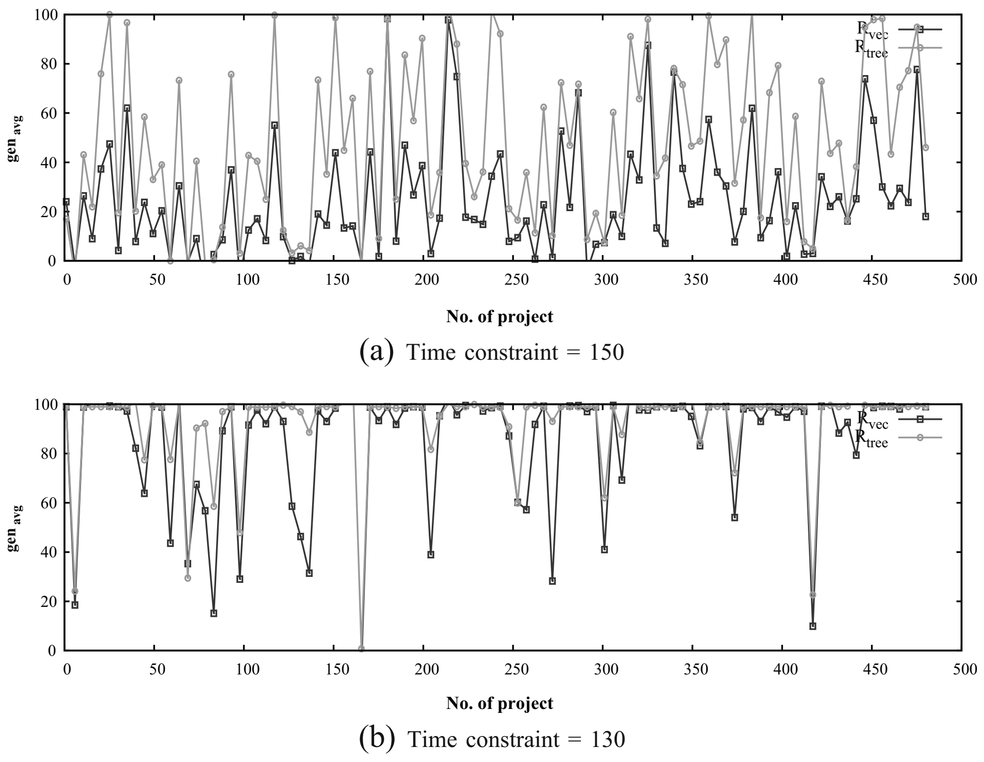


Fig. 3 Average number of generations at which evolution stopped, for R_{vec} and R_{tree} and $\Pi = 240$ with respect to the project number from PSPLIB and stronger constraints

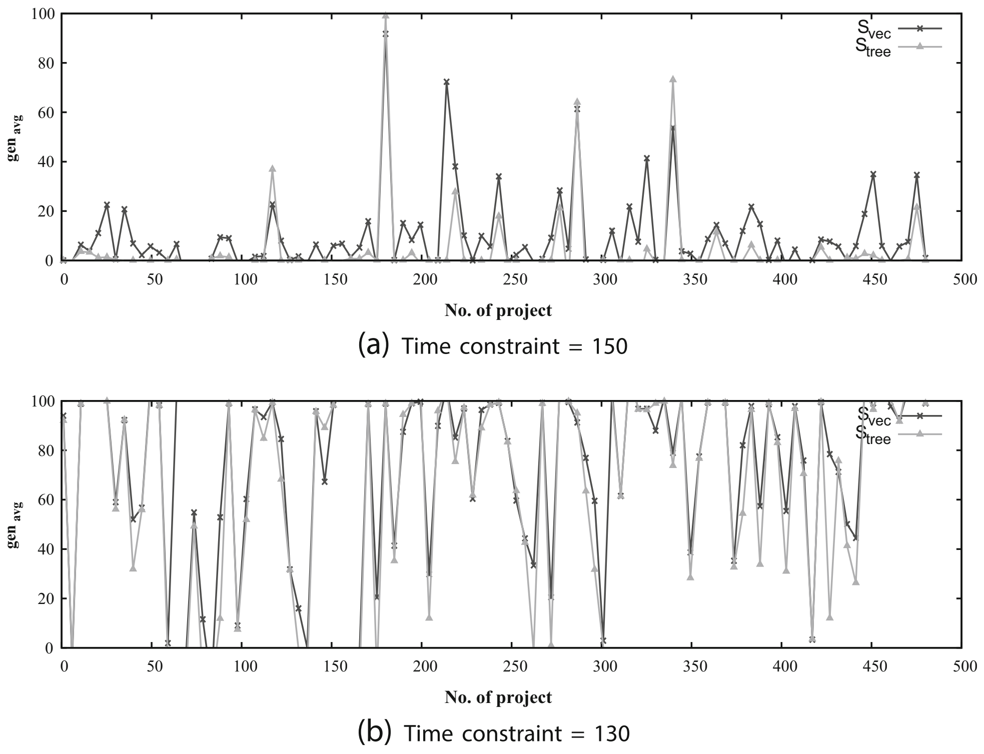
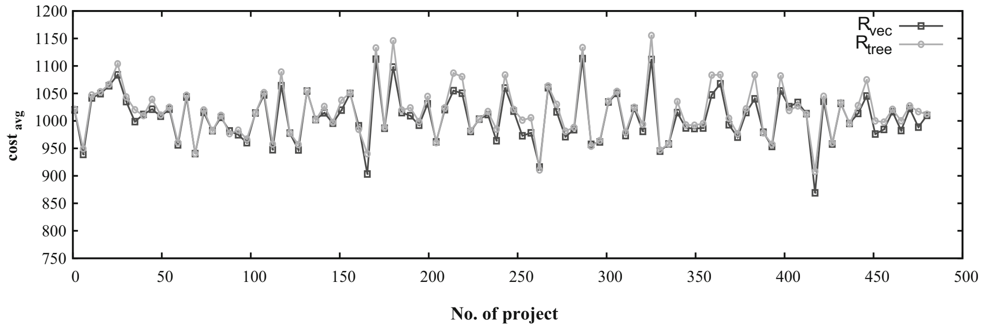
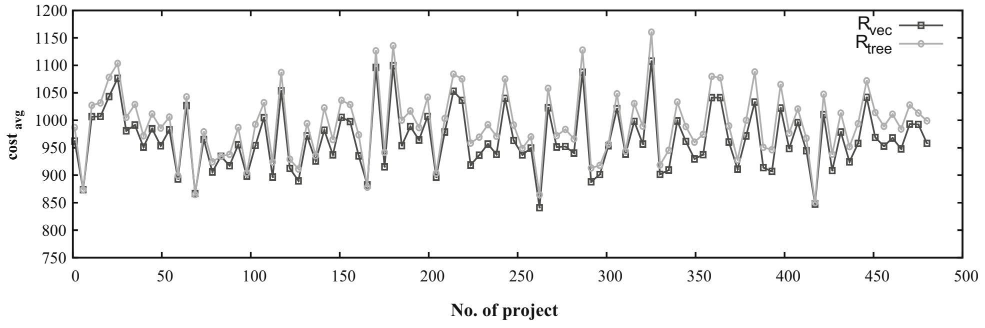


Fig. 4 Average number of generations at which evolution stopped, for S_{vec} and S_{tree} and $\Pi = 240$ with respect to the project number from PSPLIB and stronger constraints

Fig. 5 Average project cost from the best individuals in each test run, for R_{vec} and R_{tree} and $\Pi = 240$ with respect to the project number from PSPLIB and stronger constraints

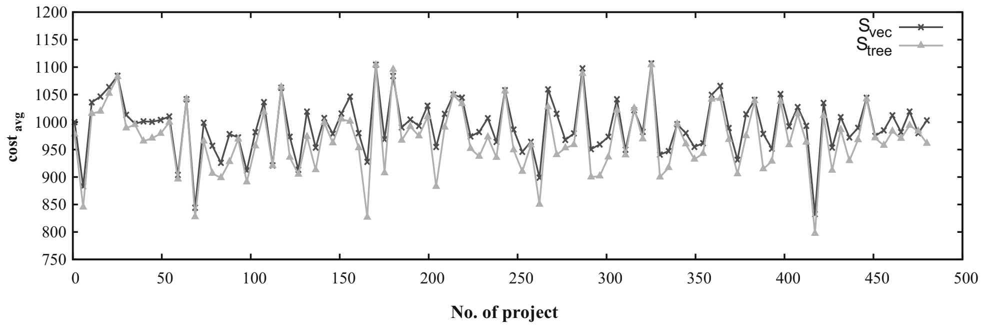


(a) Time constraint = 150

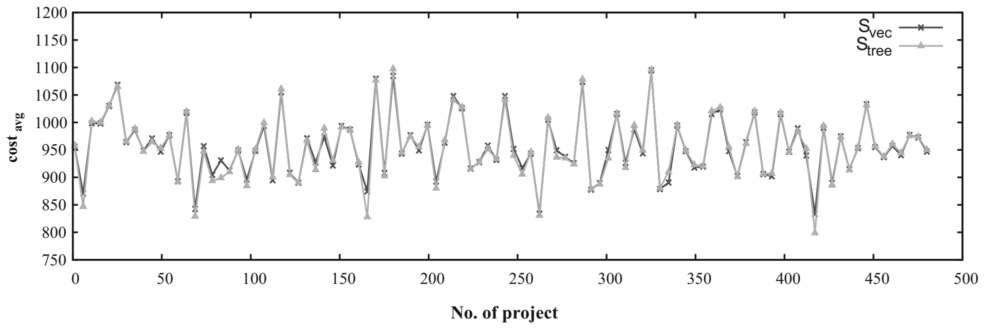


(b) Time constraint = 130

Fig. 6 Average project cost from the best individuals in each test run, for S_{vec} and S_{tree} and $\Pi = 240$ with respect to the project number from PSPLIB and stronger constraints



(a) Time constraint = 150



(b) Time constraint = 130

The characteristics of the evolutionary process concerning adjustment of the genetic operators and the size of population (Π) may be found in [33, 34]. Figure 1 shows the values of probabilities for the methods that gave the best results. Those values were chosen for further tests. It can be observed that in the vector-based methods the probability of mutation is high which makes the evolution more stochastic. Thus finding the optimum is more accidental. The SV method uses knowledge coming from strategies associated with the supervisor. Therefore, in the S_{tree} the probability of the crossover (which uses the existing genotype structures for creating new chromosomes) is more important. The crossover steadily improves features contained in the supervisors genotype whereas the mutation is used for getting out of local minima.

At the very beginning, we tested initialization of the start population in order to find out the diversity of generated individuals and the impact of the tree height on the number of proper genotype-tree structures. Next, we tested different numbers of generations.

6.1 Testing the initialization of the population

A start population in vector-based methods comprises genotypes with randomly drawn numbers (the sequence of resource numbers and decision strategies for R_{vec} and S_{vec}, respectively). Initialization of the population in tree-based methods is not so obvious. Tree structures were generated randomly by creating nodes recursively until a pre-established maximum height was reached (H). For simplicity, in our tests, each node can have one of 4 strategies (S_{tree}) or 4 resource numbers (R_{tree}), assigned with the same probability and a random location of the division, which is inversely proportional to H. Values from leaves of the tree can be assigned to each of 10 project activities, which leads to 1 048 576 possible results (4^{10}). A relationship between the height of the tree and the number of leaves is shown in Table 3. The values are obtained out of 2 mln individuals randomly generated (twice the number of possible results). After generation of the initial tree, a "replacing repair" mechanism is used to preserve the correct tree structure. The number of unique chromosomes obtained after the genotype-phenotype mapping is shown in the last column. The results show that 83 % of all possible chromosomes were generated in the initialization method, for the initial H equal 14. However, the maximum tree height does not exceed 10 with the average value of 6.15. The average number of leaves is 9.43 while 10 leaves are needed to get all possible variations. Yet, the higher the generated tree, the more

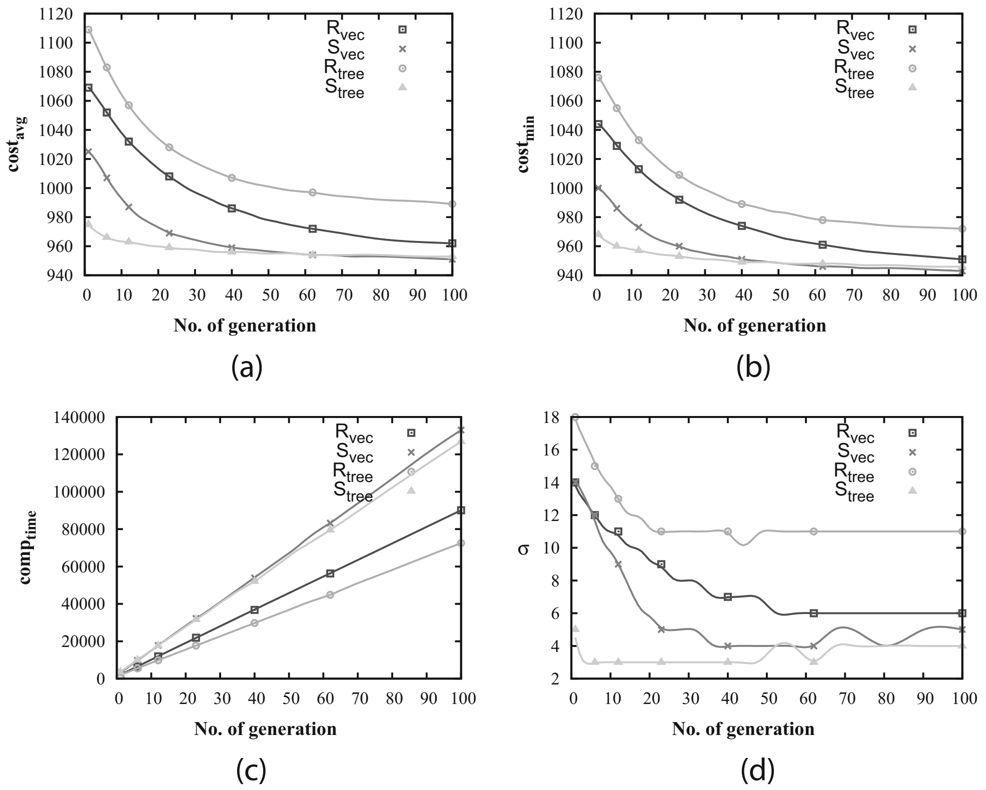


Fig. 7 Experimental results for different methods for $\Pi = 240$. min_{avg} – the average project cost from the best individuals in each test run, min – the minimal project cost, from all individuals of a given generation and all test runs, σ – the uncorrected sample standard deviation, $Comp_{time}$ – the computation time

tree leaves remain after the repair. In further tests, H is set to 10 because for bigger values the difference of the parameters has a little effect on the results. Summarizing, the initialization method can produce almost every genotype possible, which is a good start of the evolution.

6.2 Testing the number of generations

Next, we studied how many generations are needed to find a satisfactory result. A stop condition was added in order to determine which method would be better to find a result that fulfills the time constraints (duration). Figure 2 shows, that according to the definition of the fitness function (1), both the cost of the scheduled project and the makespan are minimized. With weak time constraints, the evolution of individuals does not occur because the initial population is good enough to fulfill the constraints. When the maximum acceptable project duration is decreased, additional generations are needed in the evolutionary process. The comparison of the number of generations with respect to time constraints is presented in detail in Fig. 3 for the vector-based methods and in Fig. 4 for the tree-based methods. For comparison purposes the broken line in the figures connects points obtained in the same experiment in order to expose the nature of results.

The first method, which starts to evolve individuals to improve the results is the R_{tree}. The S_{tree} for time constraint 150 still needed only a few generations, while the processing of some projects by the R_{tree} finished at

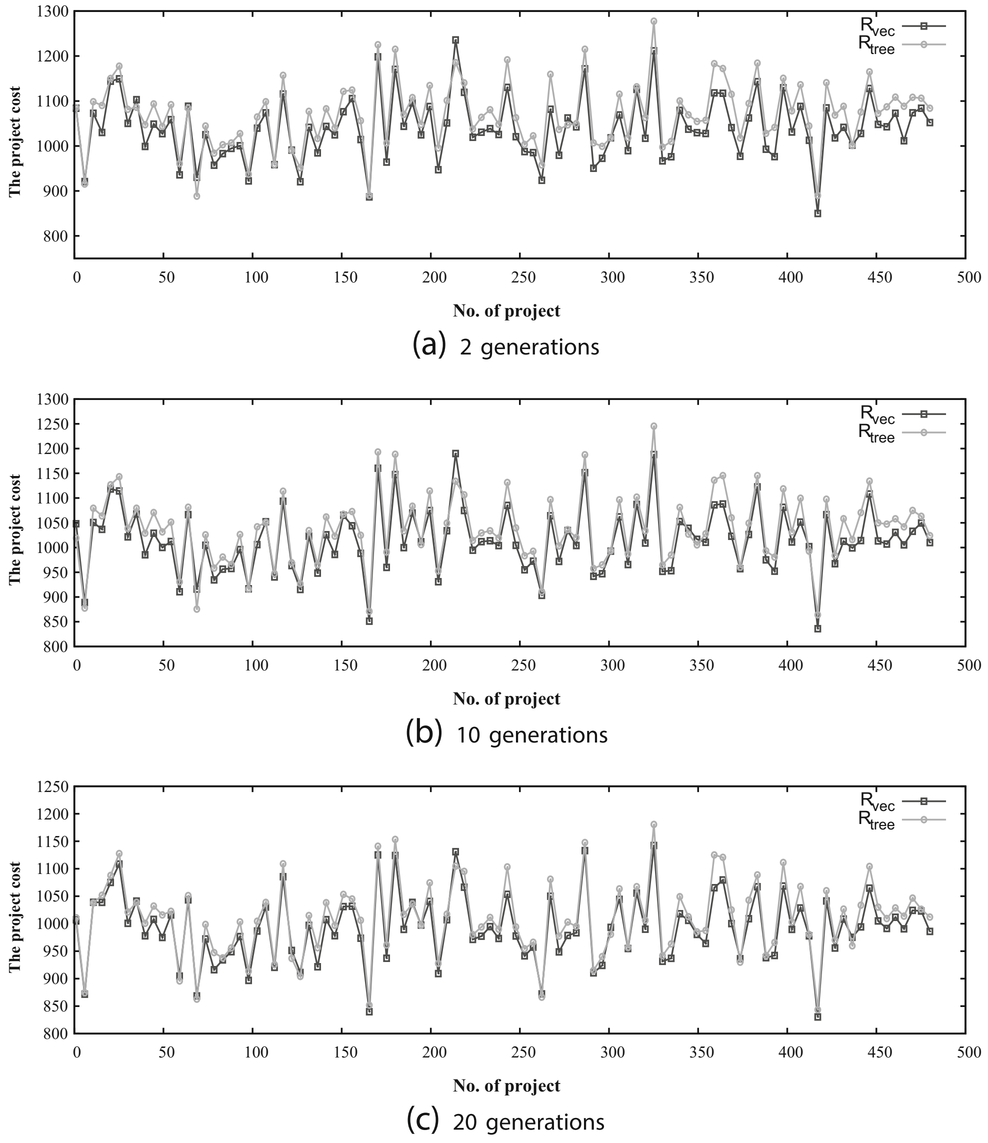


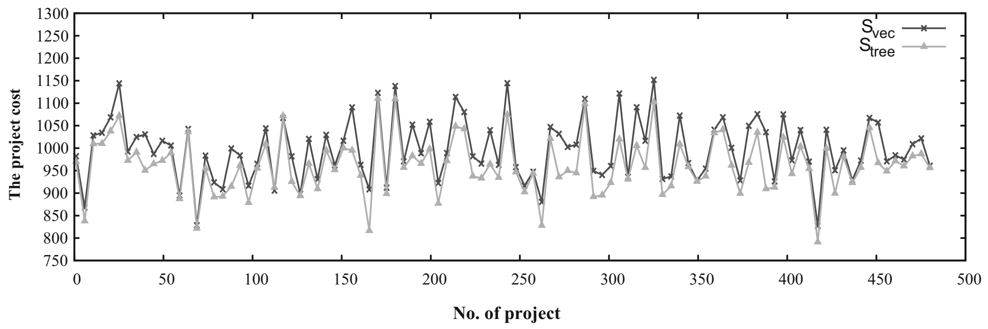
Fig. 8 The minimal project cost, from all individuals of a given generation and all test runs, for R_{vec} and R_{tree} and $\Pi = 240$ with regard to the project number from PSPLIB

the hundredth (last) generation. Reducing time constraint to 130 enforced the evolution process in each of the methods. However, the S_{tree} still needed the lowest number of generations. Also, the results were improved. The cost reduction was nearly linear for the time constraint decreased up to the point where the evolution brought no further improvement. Figures 5 and 6 show the comparison of vector-based and tree-based methods in terms of quality and time constraints. The S_{tree} gave the best results (the lowest project cost). As for the computation time, the S_{tree} was slower than the R_{vec} and the S_{vec} only when the time constraint was too strict to be fulfilled and the evolution lasted almost to the last generation.

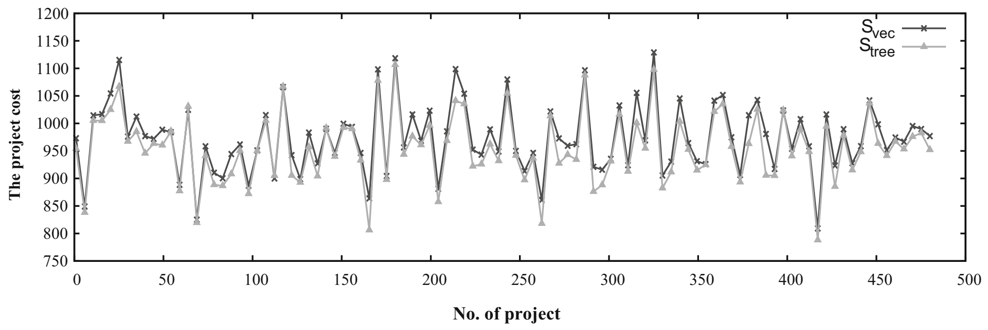
6.3 Comparative study

The performance test is executed in order to verify which method gives the best results and what the resulting costs are. At the beginning, all the methods were compared for the same sample of the project with respect to the rate of cost reduction, the complexity (computation time) and required amount of reruns (uncorrected standard sample deviation). Figure 7 shows that the SV method gives the best quality results from the very beginning of the evolution. The same average project cost from the best individuals in each population was obtained after the second generation by the S_{tree}, after the twentieth generation by the S_{vec} and thirty

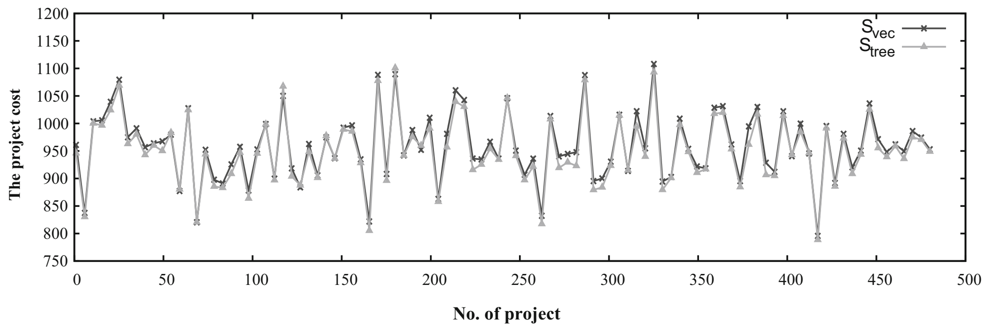
Fig. 9 The minimal project cost, from all individuals of a given generation and all test runs, for S_{vec} and S_{tree} and $\Pi = 240$ with respect to the project number from PSPLIB



(a) 2 generations



(b) 10 generations



(c) 20 generations

more generations were still needed to compensate for the difference in the quality of the results. Thus, the application of strategies makes the evolution of good quality results much faster. Project costs with regard to the number of generations are shown in detail for the vector-based methods in Figs. 8 and 9 for the tree-based methods. If we stopped the calculations after 20 generations, the S_{tree} would allow us to obtain a project cost lower by 1.4 % than the cost obtained by the S_{vec} and by 5.4 % than the cost obtained by R_{vec}. Moreover, the S_{vec} needed 40 generations (which is 2 times slower) and the R_{vec} needed 100 generations (which is 3.3 times slower) to obtain similar results. Furthermore, if we stopped after 10 generations the project cost obtained by the S_{tree} would be worse only by 0.4 % than the one obtained after 20 generations. However, in comparison with S_{vec} and the R_{vec} the result would be better by 3 % and 7.2 %, respectively. The solution with a similar cost would be obtained 2.9 times slower by the S_{vec} and 5.3 times slower by the R_{vec}. The projects containing 60 tasks are too small for practical applications. Therefore, further tests have been conducted in order to determine how fast the SV method can find good quality results for

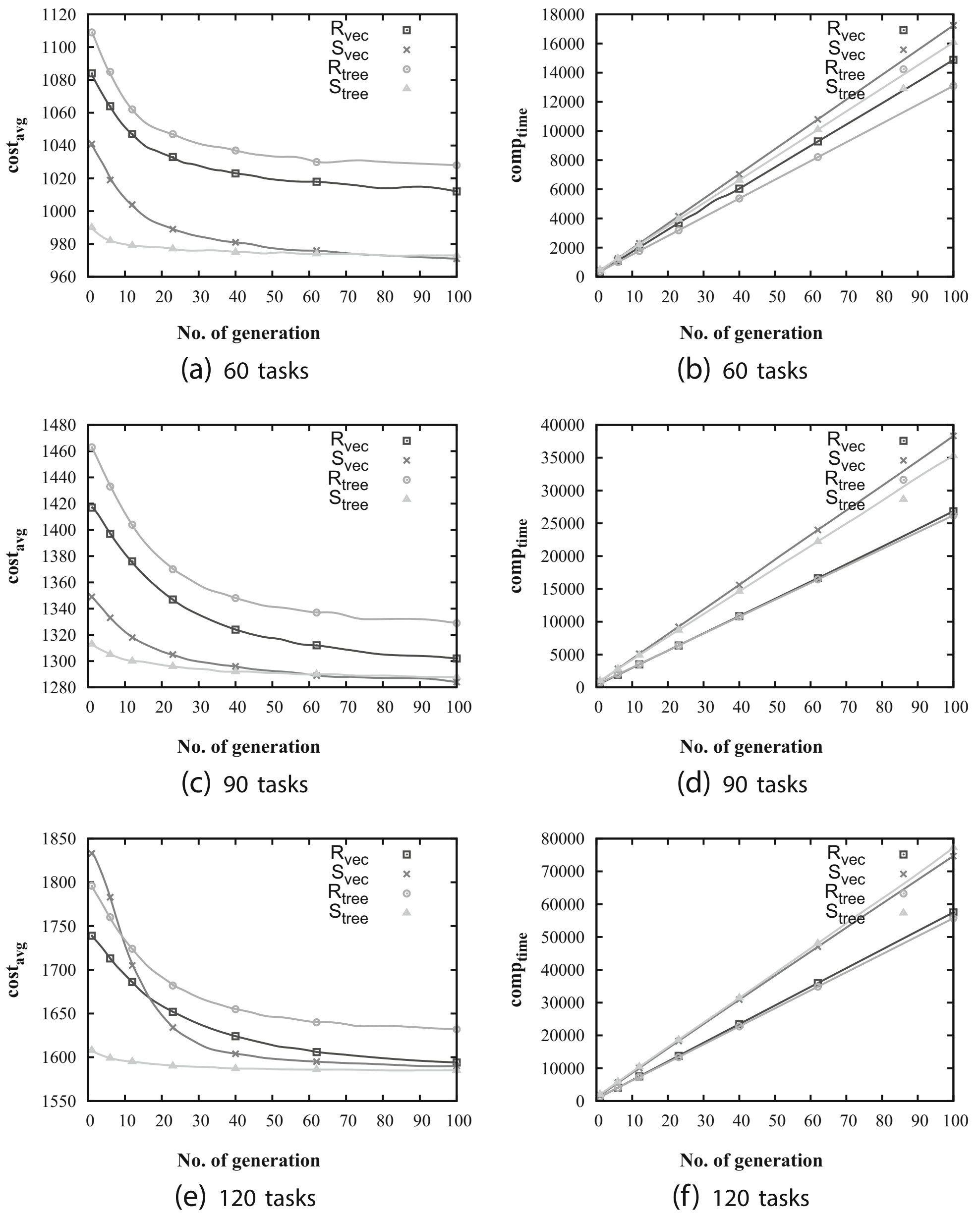


Fig. 10 Experimental results for different methods and different project complexity. $cost_{avg}$ – the average project cost from the best individuals in each test run, $comp_{time}$ – the computation time

more complex projects with a very huge space of the candidate genotypes. For that purpose, we have performed a test on 25 random projects that comprised from 60 to 120 tasks and an initial schedule with 30 extra tasks. Figure 10 shows that the S_{tree} significantly limits the search space and gives good quality results from the very beginning. The S_{tree} advantage over other methods grows with the increasing number of tasks in a project. To verify the convergence of the methods, we have calculated a point P in which the improvement of results in relation to 10 previous generations is lower than 1 %. Table 4 shows the comparison of the results in P for different methods. Only S_{vec} and S_{tree} were able to reach P in 100 generations. However, the computation time for the S_{vec} was several times greater than for the S_{tree}. If we stopped the calculations when only the S_{tree} had reached P, we would get the best quality result, (Table 5). It is worth noticing that the S_{tree} needed only 10 generations in a project with 120 tasks to obtain a result which is 6 % better than for the S_{vec}. Furthermore, the more complex the project the higher the superiority of the S_{tree} over the S_{vec}. Finally, it can be observed that in all cases computation times are nearly linear. The slopes of the lines for tree-based methods and vector-based methods are comparable.

6.4 Optimality

The final test was executed to verify whether the SV method gave the optimal results. Because the optimal results for the problem are unknown, an exhaustive search was carried out. For simplicity, only projects with 20 tasks and 3 resources with no initial schedule were tested. There are 3^{20} schedules in such projects which means that one way of executing the tasks, gives 3 486 784 401 possibilities of resource allocations. The results showed that 28 670 schedules could be calculated per second. Thus, if we wanted to check the whole search space of the projects with, for example, 30 tasks and 4 resources, we would need approximately $38.257 * 10^{10}$ days.

We tested 16 randomly generated projects that could have a maximum of 3 precedence relationships with the probability of 0.35. The values of resource parameters were set as shown in Table 6. For each project instance, 50 independent runs were performed. Table 7 shows that the optimum was found in 8 out of 16 project instances. Moreover, the best result from 50 reruns differs only by 0.14 % and the mean result was, on average, greater from the optimum one by 3.04 %. This confirms that the method can find the optimal result or the one close to it. Furthermore, the computation time of the S_{tree} was incredibly (few million times) fast when compared to the exhaustive search method.

Table 4 Comparison of results for different methods in P

No. tasks	S_{vec}	R_{vec}	S_{tree}	R_{tree}
No. generation				
60	100	90	14	80
90	72	> 100	14	> 100
120	100	> 100	10	> 100
$cost_{avg}$				
60	971.52	1015.11	979.32	1030.31
90	1288.25	–	1300.57	–
120	1590.48	–	1596.35	–
$comp_{time}$				
60	17238,39	13374,9	2527	10519,84
90	27723,98	–	5607,98	–
120	74679,81	–	8947,75	–

$cost_{avg}$ – the average project cost from the best individuals in each test run, $comp_{time}$ – the computation time

Table 5 Comparison of results for different methods in time in which S_{tree} reaches P

No. tasks	S_{vec}	R_{vec}	S_{tree}	R_{tree}	$comp_{time}$
No. generation					
60	14	16	14	18	2527
90	14	23	14	23	5603
120	12	16	10	16	8947
$cost_{avg}$					
60	999.66	1038.95	979.32	1051.91	2527
90	1315.28	1347.26	1300.57	1370.19	5603
120	1705.74	1670.62	1596.35	1705.43	8947

$cost_{avg}$ – the average project cost from the best individuals in each test run, $comp_{time}$ – the computation time

Table 6 Values of resource parameters in the experiments with 20 tasks

j	$C_e(j)$	$C_u(j)$
1	0,96	21
2	1,07	22
3	1,1	21

Table 7 Comparison of S_{tree} with optimal results, after 50 generations for $\Pi = 100$

Lp	c_{opt}	t_{opt}[h]	γ	δ	c_{min}	c_{avg}	t[s]
201	352.52	31.538	0.25	0.3	352.86	366.57	8.51
202	354.66	39.201	0.2	0.1	354.92	377.15	8.58
203	330.98	30.838	0.25	0.2	331.26	342.82	8.31
204	336.9	37.374	0.25	0.2	342.88	347.57	8.49
205	338.34	27.897	0.25	0.2	338.37	344.74	7.97
206	321.5	30.498	0.25	0.2	321.65	336.52	8.35
207	343.16	36.396	0.25	0.2	343.16	343.2	8.2
208	325.82	29.687	0.15	0.2	325.82	329.12	8.21
209	318.46	30.123	0.25	0.25	318.46	330.05	8.24
2010	408.05	37.190	0.25	0.15	408.38	424.06	9.56
2011	340.2	31.087	0.3	0.2	340.2	352.55	8.16
2012	311.52	37.994	0.25	0.3	311.55	320.51	8.26
2013	319.56	45.719	0.25	0.35	319.56	331.7	8.84
2014	316.26	30.330	0.1	0.3	316.26	328.95	7.89
2015	362.54	34.382	0.25	0.2	362.54	371.47	8.71

c_{opt} – optimal cost obtained after exhaustive search, t_{opt} – computation time of the exhaustive search, γ – probability of the crossover for which the best result was found, δ – probability of the mutation for which the best result was found, c_{min} – minimal project cost found by the SV method, c_{avg} – average project cost from the best results in each test rerun obtained by the SV method, t – computation time of S_{tree}

7 Conclusions

Experimental results have shown that since the SV needs the fewest generations; it is the fastest method that enables one to find the result that meets the time requirements. The S_{tree} extensively uses the crossover. Therefore, new individuals are mainly created from the existing genotypes, which highly reduces the search space. The mutation is used for getting out of local minima. The usage of strategies not only results in genotypes of better quality from the very beginning, but also accelerates the convergence, and thereby the cost reduction. Moreover, combining strategies and a genotype with a tree structure improves the results even more.

A core of the SV method is the development of the best genotype. To achieve this a classical construction method can be used, but only for a very limited space of feasible genotypes. Since in practice the space of the candidate genotypes can be very huge, we used evolution instead of construction. Therefore, the SV method fits in with the family of hyper-heuristics.

The main strength of the S_{tree} is that it generates quite good results for more complex projects, much faster than other genetic approaches. The comparison of the results in P showed that the computation time for the S_{vec} was several times greater than for the S_{tree}. Therefore, the SV method is especially suitable for real-time environments, where computation time is critical and a deadline for task executions is given. Moreover, the comparison of the S_{tree} with the optimal results for projects with 20 tasks and 3 resources shows that the method can find a result which is optimal or close to the optimum. On average, the best result found after 50 reruns was worse only by 0.14 % than the optimum.

Our future work will concentrate on analyzing other parameters of the evolution and different strategies for allocating resources. In general, the higher the generated tree, the more leaves remain after the repair mechanism. The number of leaves in the tree should be close to the number of activities in the project. The results showed that the generation of individuals with twice the number of possible solutions allows for obtaining 83 % of all possible genotypes, which makes a good start for the evolution process. Well-defined genotype structures and tuned genetic parameters may allow for successful adaptation of the SV method to other disciplines. The supervisor preserves some features of human supervisor. It can learn and react to dynamic changes in a system. We will check if the method can adapt to different resource requirements during the evolution. Since there is a lot of individuals in the population and the evolution lasts for many generations the process may be performed simultaneously. We will also work on the parallel DGP model in order to reduce the computation time.

References

1. Alcaraz J, Maroto C (2001) A robust genetic algorithm for resource allocation in project scheduling. Ann Oper Res 102:83–109
2. Alvarez-Valdes R, Tamarit JM (1989) Heuristic algorithms for resource-constrained project scheduling: a review and an empirical analysis. In: Advances in project scheduling, vol 9. Elsevier, Amsterdam, pp 113–134
3. Banzhaf W (1994) Genotype-phenotype-mapping and neutral variationa case study in genetic programming. In: Parallel problem solving from naturePPSN III. Springer, pp 322–332
4. Bouleimen K, Lecocq H (2003) A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. Eur J Oper Res 149(2):268–281. http://linkinghub.elsevier.com/retrieve/pii/S0377221702007610
5. Deiranlou M, Jolai F (2009) A new efficient genetic algorithm for project scheduling under resource constrains. World Appl Sci J 7(8):987–997
6. Demeulemeester EL, Herroelen WS (1996) Optimal procedures for the discrete time/cost trade-off problem in project networks. Eur J Oper Res 88(1):50–68
7. Demeulemeester EL, Herroelen WS (1997) New benchmark results for the resource-constrained project scheduling problem. Manag Sci 43(11):1485–1492
8. Deniziak S, Ciopiński L, Pawiński G (2016) Design of real-time computer-based systems using developmental genetic programming. In: Handbook of genetic programming appliactions. Springer, in print
9. Deniziak S, Gorski A (2008) Hardware/software co-synthesis of distributed embedded systems using genetic programming. In: Evolvable systems: from biology to hardware. Springer, pp 83–93
10. Elbeltagi E, Hegazy T, Grierson D (2005) Comparison among five evolutionary-based optimization algorithms. Adv Eng Info 19(1):43–53
11. Erenguc SS, Ahn T, Conway DG (2001) The resource constrained project scheduling problem with multiple crashable modes: an exact solution method. Naval Res Logist (NRL) 48(2):107–127
12. Fang HL, Ross P, Corne D (1993) A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. University of Edinburgh, Department of Artificial Intelligence
13. Frankola T, Golub M, Jakobovic D (2008) Evolutionary algorithms for the resource constrained scheduling problem. In: Proceedings of 30th international conference on information technology interfaces, vol 7269. Information Technology Interfaces, p 715–722
14. Hendrickson C, Au T (2008) Project management for construction: Fundamental concepts for owners, engineers, architects, and builders. Chris Hendrickson
15. Hindelang T, Muth J (1997) A dynamic programming algorithm for decision cpm networks. Oper Res 27(2):225–241
16. Keller RE, Banzhaf W (1996) Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In: Proceedings of the 1st annual conference on genetic programming. MIT Press, pp 116–122
17. Keller RE, Banzhaf W (1999) The evolution of genetic code in genetic programing. In: Proceedings of the genetic and evolutionary computation conference (GECCO 1999). Information Technology Interfaces, pp 1077–1082
18. Klein R (2000) Resource-constrained scheduling problems. Springer
19. Kolisch R (1996) Serial and parallel resource-constrained project scheduling methods revisited: theory and computation. Eur J Oper Res 90(2):320–333
20. Kolisch R, Hartmann S (2006) Experimental investigation of heuristics for resource–constrained project scheduling: an update. Eur J Oper Res 174:23–37
21. Kolish R, Sprecher A (1996) Psplib a project scheduling library. Eur J Oper Res 96:205–216
22. Koulinas G, Anagnostopoulos K (2013) A new tabu search-based hyper-heuristic algorithm for solving construction leveling problems with limited resource availabilities. Autom Construct 31:169–175
23. Koza J, Keane MA, Streeter MJ, Mydlowec W, Yu J, Lanza G (2003) Genetic programming IV: routine humancompetitive machine intelligence. Kluwer Academic Publishers
24. Koza JR (1992) Genetic programming: on the programming of computers by means of natural selection. MIT Press, Cambridge
25. Koza JR (2010) Human–competitive results produced by genetic programming. Gen Program Evolv Mach 11:251–284
26. Koza JR, Poli R (2005) Search methodologies: introductory tutorials in optimization and decision support techniques. Springer, New York
27. Merkle D, Middendorf M, Schmeck H (2002) Ant colony optimization for resource-constrained project scheduling. IEEE Trans Evol Comput 6(4):333–346
28. Mhring RH, Shulz AS, Stork F, Utez M (2003) Solving project scheduling problems by minimum cut computations. Manag Sci 49(3):330–350
29. Mohan BC, Baskaran R (2012) A survey: ant colony optimization based recent research and implementation on several engineering domain. Expert Syst Appl 39(4):4618–4627
30. Orvosh D, Davis L (1993) Shall we repair? genetic algorithmscombinatorial optimizationand feasibility constraints. In: Proceedings of the 5th international conference on genetic algorithms. Morgan Kaufmann Publishers Inc, p 650
31. Patterson J, Harvey R (1997) An implicit enumeration algorithm for the time/cost tradeoff problem in project network analysis. Found Control Eng 4(2):107–117
32. Pawiński G, Sapiecha K (2014) Cost-efficient project management based on critical chain method with partial availability of resources. Control Cybern 43(1)
33. Pawiński G, Sapiecha K (2014) A developmental genetic approach to the cost/time trade-off in resource constrained project scheduling. In: 2014 Federated conference on computer science and information systems (FedCSIS). IEEE, pp 171–179
34. Pawiński G, Sapiecha K (2016) An efficient solution of the resource constrained project scheduling problem based on an adaptation of the developmental genetic programming. In: Recent advances in computational optimization. Springer, pp 205–223
35. Pinedo M, Chao X (1999) Operations scheduling with applications in manufacturing, 2nd edn. Irwin/McGraw–Hill, Boston
36. Sapiecha K, Ciopiński L, Deniziak S (2016) Synthesis of self-adaptive supervisors of multi-task real-time object-oriented systems using developmental genetic programming. In: Recent advances in computational optimization. Springer, pp 55–74

37. Sapiecha K, Ciopinski L, Deniziak S (2014) An application of developmental genetic programming for automatic creation of supervisors of multi-task real-time object-oriented systems. In: 2014 Federated conference on computer science and information systems (FedCSIS). IEEE, pp 501–509
38. Vanhoucke M (2005) New computational results for the discrete time/cost tradeoff problem with time-switch constraints. Eur J Oper Res 165(2):359–374
39. Weise T (2009) Global optimization algorithms-theory and application. Self-Published
40. Wilson G, Heywood M (2006) Probabilistic adaptive mapping developmental genetic programming (pam dgp): a new developmental approach. In: Parallel problem solving from nature-PPSN IX. Springer, pp 751–760
41. Zhang H, Li X, Li H, Huang F (2005) Particle swarm optimization-based schemes for resource-constrained project scheduling. Autom Construct 14(3):393–404