Proceedings of the 2014 Federated Conference on Computer Science and Information Systems pp. 171–179
DOI: 10.15439/2014F151
ACSIS, Vol. 2

A Developmental Genetic Approach to the cost/time trade-off in Resource Constrained Project Scheduling

Grzegorz Pawiński
Department of Computer Science
Kielce University of Technology
al.1000-lecia P.P. 7
25-314 Kielce, Poland
Email: g.pawinski@tu.kielce.pl

Krzysztof Sapiecha
Faculty of Electrical and Computer Engineering
Cracow University of Technology
ul. Warszawska 24
31-155 Kraków, Poland
Email: krzysztof.sapiecha@gmail.com

***Abstract*—In this paper, the use of Developmental Genetic Programming (DGP) for solving a new extension of the Resource-Constrained Project Scheduling Problem (RCPSP) is investigated. We consider a variant of the problem when resources are only partially available and a deadline is given but it is the cost of the project that should be minimized. RCPSP is a well-known NP-hard problem but in its original formulation it does not take into consideration initial resource workload and it minimises the makespan. Unlike other genetic approaches, where genotypes represent solutions, a genotype in DGP is a procedure that constructs a solution to the problem. Genotypes (the search space) and phenotypes (the solution space) are distinguished and a genotype-to-phenotype mapping (GPM) is used. Thus, genotypes are evolved without any restrictions and the whole search space is explored. The goal of the evolution is to find a procedure constructing the best solution of the problem for which the cost of the project is minimal. The paper presents genetic operators as well as GPM specified for the DGP. Experimental results showed that our approach gives significantly better results compared with methods presented in the literature.**

I. Introduction

THE Resource-Constrained Project Scheduling Problem (RCPSP), attempts to reschedule project tasks efficiently using limited renewable resources minimising the maximal completion time of all activities [Brucker et al., 1998], [Demeulemeester and Herroelen, 1997], [Demeulemeester and Herroelen, 2002]. A single project consists of m tasks which are precedence-related by finish-start relationships with zero time lags. The relationship means that all predecessors have to be finished before a task can be started. To be processed, each activity requires a limited resource R which is unique and therefore it has to perform different activities sequentially.

RCPSP is an NP-complete problem which is computationally very hard [Blazewicz et al., 1983]. The RCPSP occurs frequently, in high scale project management such as software development, power plant building, and military industry projects such as design, development and building of nuclear submarines [Pinedo and Chao, 1999]. The authors in [Möhring et al., 2003] state that it is one of the hardest problems of Operational Research. Results of the investigation in [Hartmann and Briskorn, 2010] showed that the best performing heuristics for solving the RCPSP were the Genetic algorithm (GA) of Hartmann [Hartmann, 1998] and the Tabu search (TS) procedure of Bouleimen and Lecocq [Bouleimen and Lecocq, 2003].

Various RCPSP extensions have been developed for solving practical problems [Hartmann and Briskorn, 2010]. However, there is still free room for research. In this paper we will attack the RCPSP where resources have already got their own schedule (like in a software house). Such tasks cannot be moved. Hence, the resources are available only in particular time periods. The goal is to allocate resources to the project tasks, taking into consideration the availability of resources, in order to minimise the total cost of the project and complete it before a deadline.

The resources can be used only in specified time periods what makes the problem computationally even more complex. To overcome this complexity Genetic programming (GP) [Koza, 1992] will be adopted. GP is a domain-independent method that genetically breeds a population of computer programs to solve a problem. It is an extension of the GA [Hartmann, 1998], [Goldberg, 1989], in which the structures in the population are not fixed-length character strings that encode candidate solutions to a problem, but programs that, when executed, produces the candidate solutions to the problem [Koza and Poli, 2005]. Most GP 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. Developmental genetic programming (DGP) [Koza et al., 2003] makes a distinction between the search space and the solution space. DGP evolves a schedule construction procedure instead of the schedule itself. Thus, genotypes are evolved without any restrictions and the whole search space is explored. The quality of the solution is evaluated on the phenotype after genotype-to-phenotype mapping (GPM). The mapping is critical to the performance of the search process [Keller and Banzhaf, 1999].

To summarize, the purpose of the paper is to introduce a new DGP approach for solving RCPSP when resources are

978-83-60810-58-3/$25.00

partially available. The next section of the paper contains a brief overview of related work. A motivation to the research is given in section 3. Section 4 presents an idea of the adaptation of developmental genetic programming for a solution of the RCPSP. In section 5, computational experiments to evaluate our approach and a comparison with other methods are given. The paper ends with conclusions.

II. Related work and preliminary remarks

Both exact and heuristic methods have been used for solving the RCPSP. Among the first ones, a depth-first branching scheme with dominance criteria and the bounding rules was proposed [Demeulemeester and Herroelen, 1997]. In [Brucker et al., 1998] a branching scheme which starts from a graph representing a set of conjunctions and disjunctions was used. Another method, a tree search algorithm was presented in [Mingozzi et al., 1998]. It is based on a mathematical formulation that uses lower bounds and dominance criteria. However, heuristics have been preferred instead of exact methods due to substantial limitations of these latter ones. In-depth study of the performance of recent RCPSP heuristics can be found in [Kolisch and Hartmann, 2006]. Heuristics described by the authors, include X-pass methods, also known as priority rule based heuristics, classical metaheuristics, such as Genetic algorithms, Tabu search, Simulated annealing (SA), and Ant Colony Optimisation (ACO). They give a performance comparison of these methods as applied to different standard instances sets, generated by ProGen in the PSPLIB [Kolish and Sprecher, 1996]. Two approaches of Tabu Search, for artificially created dataset instances, but based on real-world instances (got from Volvo IT and verified by experienced project manager), were investigated in [Skowronski et al., 2013].

One of the latest review papers on solving RCPSP by exact methods and heuristics may be found in [Deiranlou and Jolai, 2009], where a particular attention was paid to GAs. The authors introduced a new crossover operator and auto-tuning for adjusting the rates of crossover and mutation operators. In [Zamani, 2013], an effective hybrid evolutionary search method which integrates a genetic algorithm with a local search was presented. Two approaches for solving the problem with GAs and GP are given in [Frankola et al., 2008]. The authors achieved good quality results by the use of GAs. With GP, they described a methodology to evolve scheduling heuristics in a small amount of time. DGP [Keller and Banzhaf, 1999], [Koza et al., 2003] is an adaptation of GP [Koza, 1992] to the optimisation problems. The DGP is quite new (from 1999) and has never been applied to the RCPSP. However, it has already been successfully applied in the design of electronic circuits, control algorithms [Koza et al., 2003], strategy algorithms in computer games, the synthesis of embedded systems [Deniziak and Górski, 2008], etc. Many of the human-competitive results that were produced using runs of genetic programming that employed a developmental process are described in [Koza, 2010]. Reinforcement Learning (RL) is another machine learning algorithm that was used for solving the RCPSP. RL determines of how software agents ought to take actions so as to achieve one or more goals. The learning process takes place through interaction with the JABAT environment [Jedrzejowicz and Ratajczak-Ropel, 2014].

According to [Alcaraz and Maroto, 2001], the optimal solution can be achieved by exact procedures only for small projects, usually containing less than 60 tasks and 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 would be desired to enhance the process. For many problems, restrictions are imposed on how the structure of genotype may be created. GP algorithms handle the problems by constrained genetic operators in the manner, which makes them produce only legal individuals. The method achieved respectable results for the generation of efficient programs in different domains, e.g. mathematical calculations, robot control, text recognition, etc. In 36 cases, obtained results were as good as or even better then known solutions [Koza and Poli, 2005]. However, constrained operators create infeasible regions in the search space, also eliminating sequences of genes which may lead to high quality solutions. In the DGP approach the problem does not exists anyway. Because of separating the search space from the solution space, legal as well as illegal genotypes are evolved, while each genotype is mapped onto a legal phenotype. It is worth to notice that the evolution of an illegal genotype may lead to the legal genotype constructing the expected result. Thus, the whole search space is explored.

III. Motivation

Classical RCPSP as well as its extensions presented in the literature do not encompass some practical problems. In the classical approach the goal of optimisation is to minimise the makespan without taking into consideration the project cost. Moreover, resources are assumed to be steadily available during the execution of the whole project. In spite of that, developers in a software house or resources of an enterprise building houses, for example, may have initial workloads when starting a new project. A goal in such cases is a cost/time trade-off, i.e. to minimize the total cost while satisfying time constraints. Therefore, an extension of RCPSP where resources have already got their own schedules and cost/time trade-offs are dominant is necessary. The problems have been addressed in the literature [Ahn and Erenguc, 1998], [Drexl et al., 2000], but not combined. Together, they make the RCPSP computationally very complex. To our best knowledge there was no attempt to deal with the problem. Satisfactory results in the matter will make it possible to efficiently solve more complex real life problems faster and better, for which there currently is not any sufficient solution.

IV. Adaptation of the DGP for the RCPSP

In the DGP genotype and phenotype are distinguished. A genotype in classical GAs represents a target solution, while

in DGP the genotype comprises a procedure that construct a solution of the problem. So, if the target solution (phenotype) is a sequence of tasks with allocated resources, which is usually created by the project manager, then the construction of the solution will be a method of how the project manager selects a resource to allocate for each of the tasks. Therefore, DGP does not evolve a project schedule but the project manager itself. A genotype defines how the project manager uses resource allocation strategies to create a project schedule. During evolution only genotypes are evolved, while the genotype-to-phenotype mapping is used to create phenotypes. In that way, the quality of the phenotype may be evaluated in order to find procedures constructing the best solution.

An evolution process in DGP is similar to other genetic approaches. It starts with an initial population with POP individuals. Subsequently, pairs of individuals are randomly drawn from the population and subject to the operation of crossover and mutation. Then, the fitness of newly created genotypes is calculated. If they satisfy time constraints, they pass the *life test* and are added to the current population. Thus, the population size in each generation is at most $2 \cdot POP$. Finally, the reproduction operator is used POP times to reduce the population to its former size. It selects the best individuals for the new population that replaces the current one. This iterative process is repeated over many generations until a predefined number of generations (GEN) has been reached.

A. Genotypes and phenotypes

In our method a genotype of the project manager is represented by a binary tree that comprises resource allocation strategies and a way of applying them for the activities. The tree edges represent a division of the list of activities into two subgroups, while nodes specify a location of the division ($node_d$) and a resource allocation strategy ($node_s$) (Figure 1). The strategy, which will be assigned to each of the subgroups is specified in an appropriate child of the node. A resource may be assigned according to one of the following strategies:

1) is the fastest,
2) is the cheapest,
3) allows to start the task as soon as possible,
4) allows to finish the task as soon as possible,
5) causes the smallest increase of the project time,
6) causes the smallest increase of the project cost.

The initial population consists of individuals generated randomly by recursively creating nodes until a pre-established maximum $Tree_{height}$ is reached. An increase of the tree height causes doubling the divisions and hence the number of leafs of the tree. Therefore, in order to get all possible variations of strategies, the number of the leafs (1) should be at least the number of activities in the project. At the beginning, the genotype is a full tree, where each node has one of the strategies assigned with the same probability and a random $node_d$, which is inversely proportional to the tree height. However, it has to be verified whether nodes contain improper values of $node_d$. The dividing point cannot be bigger than the number of activities in currently considered subgroup. One of the repairing mechanisms could be a "deleting repair" that removes all children of the invalid node. The process is similar to withering of unused features in live organisms, like in intron splicing [Watson et al., 1992]. But we used a "replacing repair" that replaces the invalid node by any of its children, instead of removing the entire branch. Therefore, more genetic information will be kept in the genotype.

Fig. 1. Tree node, where $node_d$ - dividing point, $node_s$ - decision strategy

$$Leafs_{num} = 2^{Tree_{height}-1} \tag{1}$$

B. Genotype-to-phenotype mapping

A Genotype-to-phenotype mapping (GPM) is used to transform the tree structure into a sequence of decision strategies, corresponding to the project activities. The procedure is shown on Algorithm 1. GPM is done by traversing the tree in the depth-first order starting from the top node (the root). If a node has children, a $node_d$ is used for dividing currently considered project activities into two subgroups and strategies are assigned to them. The left child defines a strategy for the first subgroup (from the first to $node_d$ task in the currently considered group of activities) and the right child for the other (from $node_d$ task to the last task in the currently considered group of activities) (Figure 2a). Then, subgroups of activities are passed to offspring nodes and the process is continued (Figure 2b). As a result, we obtain a sequence of decision strategies (*strat*), each corresponding to the given project activity (Figure 2c).

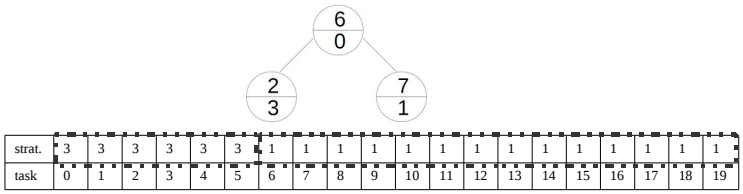
Subsequently, the decision strategies are used to create a project schedule with assigned resources (phenotype). To this end, the following steps have to be carried out:

1) search the project's activities, according to the precedence relationships, in order to find a list of ready-to-start activities,
2) assign the strategies with corresponding activities and execute the strategy to calculate a resource to allocate,
3) calculate a start time for each activity, based on the earliest precedence relationships and the feasible time of a resource,
4) repeat from step 1) until there are activities that are still unassigned.

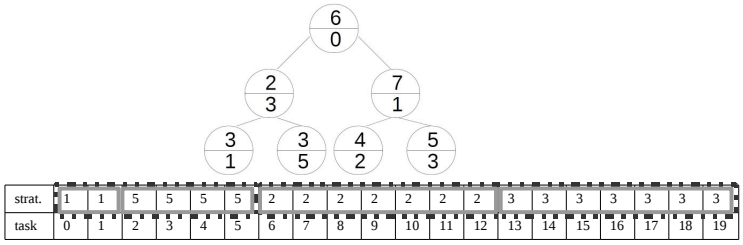
Finally, a feasible project schedule is obtained, which is used for calculating the genotype fitness. It is worth to notice that a genotype is always mapped onto a legal phenotype.

C. Fitness function

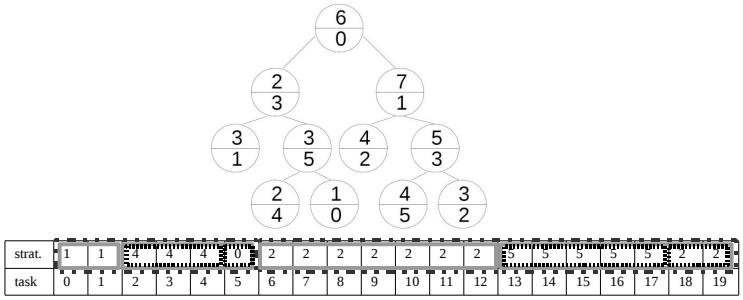
Each individual in the population is measured in order to check the quality of the solution. A numerical value, called fitness, is calculated for the project schedule obtained after



(a) II division



(b) III division



(c) IV division

Fig. 2. Genotype with 6 decision strategies (0-5) represented by a binary tree ($Tree_{height} = 4$) and a sequence of decision strategies (strat.) after the decoding, corresponding to the project activities (task)

Algorithm 1 The procedure of genotype-to-phenotype mapping, where $node$ - currently considered node, i - the index of activity in the list of activities, $size$ - the number of activities in the currently considered group of activities, $strat$ - a sequence of decision strategies, each corresponding to the given project activity

DECODETREE($root, 0, No.of activities$)
function DECODETREE($node$, i, $size$)
  if $node \neq NULL$ **then**
    DECODETREE($left\ child, i, node_d$)
    if $node = Leaf$ **then**
      $strat[i, i + size - 1] \leftarrow node_s$
    end if
    DECODETREE($right\ child, i+node_d, size-node_d$)
  end if
end function

the genotype-to-phenotype mapping. It is defined as the project cost (C), using the following equation:

$$C = \sum_{j=1}^{r} C_e(j) \cdot T_p + \sum_{j=1}^{n} \left(C_u(j) + \sum_{i=1}^{m} T(i,j) \cdot C_e(j) \right) \quad (2)$$

where $C_e(j)$ - cost of task execution per time unit by the resource R_j, T_p - the project duration, $C_u(j)$ - resource R_j unit cost, $T(i,j)$ - safe time estimate of task i being executed by the resource R_j, n - the number of resources used in the project schedule, m - the number of tasks, r - the number of resources in the resource library. The first sum corresponds to the resource maintenance cost in the project. In the second sum, the first element is the resource deployment cost and the second is the cost of the task's execution. The algorithm is set to find an individual with minimal fitness, meaning that genotypes that produce lower project cost are considered to be better ones.

D. Genetic operators

The genetic operations that are performed during the run (i.e. crossover, mutation, reproduction) are based on techniques described in [Deniziak and Wieczorek, 2012]. A crossover is applied with the probability $P_{cross} \in [0, 1]$ on a randomly selected pairs of individuals. There are at most $\frac{POP}{2}$ such pairs. Next, the decision trees in pairs are pruned by removing a randomly selected edge. Then, subtrees are swapped between both parent genotypes. Similarly, a mutation is applied on each genotype in the population with the

probability $P_{mut} \in [0, 1]$. Afterwards, one of the following modifications, selected with the same probability, is done on the decision tree:

1) a randomly selected node is changed to another,
2) a randomly selected edge is pruned and the subtree is removed,
3) two random nodes are created and added to a randomly selected leaf.

Modifications are done only when the subtree contains more than one node. If the newly created tree is too high, it is pruned to the allowed height. Then, faulty nodes are removed to preserve the correct tree decoding. Implementation of genetic operators ensures that the correct genotype-tree structure is always kept. Moreover, they neither break precedence constraints nor produce infeasible schedules.

Reproduction copies the best individuals from the current generation to the next generation. We have tested several variants of reproduction such as the ranking method, proportional selection (roulette-wheel selection) and tournament selection. In the tournament selection some number of individuals (called a tournament size TS_{size}) are randomly chosen from the population and a genotype with the lowest fitness is selected. The chance of the individual's being selected in the roulette-wheel method is inversely proportional to its fitness. It is similar in ranking selection, but selection probabilities depend on an individual's position in the ranking. Each individual in the population has a numerical rank based on its fitness.

V. Experimental results

A. Test Instances

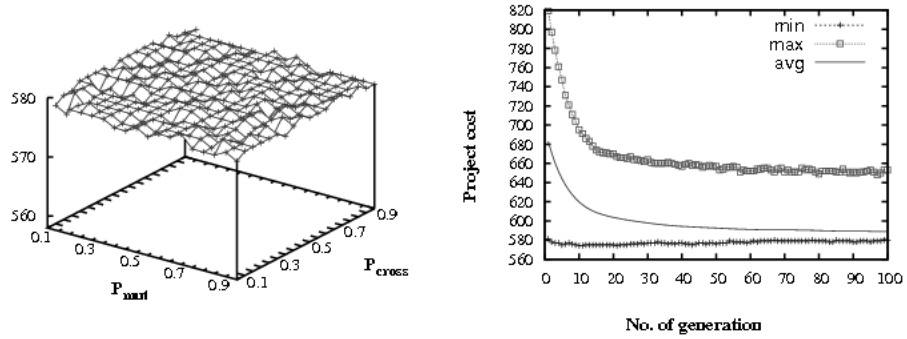
The algorithm described in the paper was tested on projects from PSPLIB, developed by [Kolish and Sprecher, 1996].The library for RCPSP contains 2040 projects with 30, 60, 90 and 120 activities for which either optimal, best-known or lower bound solutions are given. For each problem size, a set consists of 480 instances in groups of ten, which have been systematically generated by varying three parameters: network complexity, resource factor, and resource strength. The parameters have a big impact on the hardness of the project instances [Kolish and Sprecher, 1996]. The set with 30 non-dummy activities is the hardest standard set of RCPSP-instances for which all optimal solutions are currently known [Demeulemeester and Herroelen, 1997]. In our study we used project instances with 30 non-dummy activities. The renewable resources were randomly generated such that the resource development cost $C_u(j)$ and the cost of a activity's execution $C_e(j)$ might vary up to 10% from default values, which were 20 and 1 respectively. They are general purpose resources, so they may execute any of the activities. A single group of the project instances was examined, in which 10 independent runs were performed and the results were averaged. However, we considered an extension of DTCTP where resources have already got their own schedule (initial schedule). To this end, a project instance was randomly drawn from the same group in the PSPLIB. Then, activities from the project were randomly allocated to resources. Such activities cannot be moved and therefore the resources were available only in particular time periods. So even though we take the project instances from PSPLIB, our results cannot be compared with optimal because of a different problem statement. In most tests, the population size POP = 30, the number of generations (GEN) was set to 100 and the Tree height was set to 10, because for bigger values the difference of population sizes has little effect on the results.

B. Main results

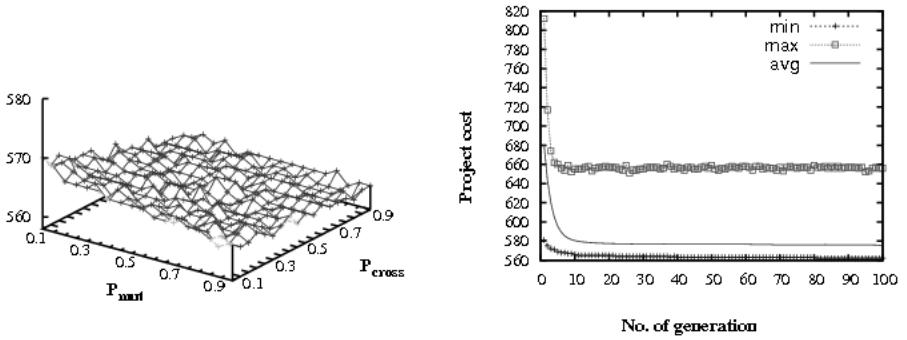
1) The selection method test: At first, we have tested three reproduction methods: ranking, roulette-wheel and tournament selection ($TS_{size} = 3$). The Figures 3 and 4 present the project cost averaged from 100 project schedules. In the roulette-wheel method (Figure 3a), the cost after 100 generations is approximately the same for various probabilities of mutation and crossover. The population contains randomly selected individuals with the probability proportional to their fitness and with no certainty that the best one will be reproduced. At the beginning a population is the most varied and its diversity lowers in further generations. But the best result in every subsequent generation is getting worse. The project cost is almost the same, along with the increase of the probability of both genetic operators, while in other methods it is significantly lower. In other methods the best results were obtained for $P_{cross} + P_{mut} > 1$. In the tournament method (Figure 2b) and ranking method (Figure 3c) there are much more good quality results in generations than bad ones. Good quality results start to dominate in the population very quickly along with the increasing number of generations and therefore the project cost decreases. The convergence of the ranking method is slightly faster than the tournament method. In the former one the rapid fall of the average cost stops after 5 generations while in the latter after 9 generations. Further improvement is very slight, but it occurs till the last generation. However, the best results were obtained for the tournament selection and therefore it was chosen for further examination.

2) Test of various probabilities of genetic operators: Next, tests were executed in order to examine how various probabilities of mutation (Figure 4a and 4b) and crossover (Figure 4c and 4d) influence the algorithm performance. The Figures show the lowest project cost in generations. Usually, the project cost becomes lower along with increasing P_{cross} as well as increasing P_{mut}, because the operators produce more new genotypes and the population is more varied. Thus, the chance of finding the optimal solution is bigger. However, only the best genotypes will be selected to the next generation. The variety of individuals may also be increased by increasing their number in generations. Generally, if the POP is bigger, the results are improved. Yet, the slope of the project cost reduction is similar.

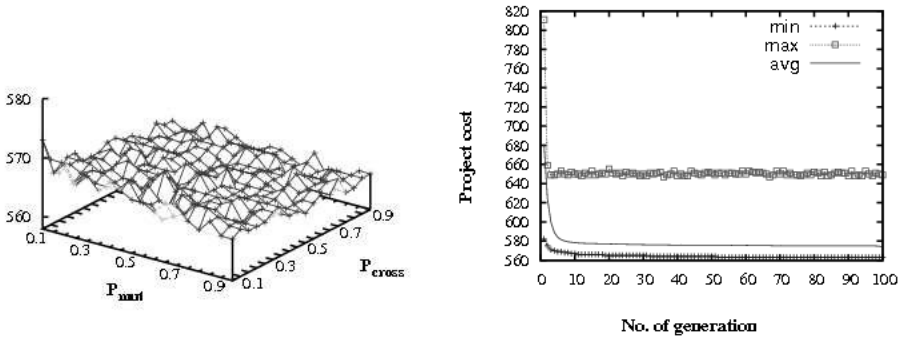
Further tests were performed to study the influence of project time constraints on the final result. Usually, the longer the project time allowed, the more genotypes were passing the *life test* so the population in each generation was bigger



(a) roulette-wheel selection ($POP_{size} = 30$, $Tree_{height} = 10$)



(b) tournament selection ($TS_{size} = 3$, $POP_{size} = 30$, $Tree_{height} = 10$)

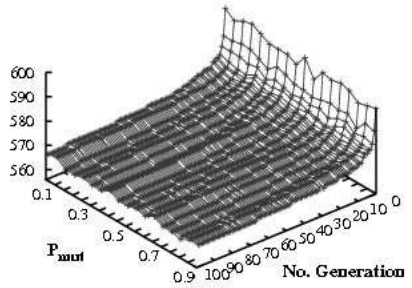


(c) ranking selection ($POP_{size} = 30$, $Tree_{height} = 10$)

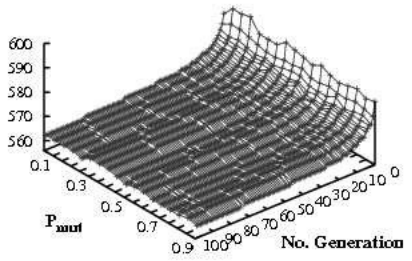
Fig. 3. Minimal cost of the project after 100 generations (left column) and the project cost in generations for $P_{cross} = 0.8$, $P_{mut} = 0.8$ (right column), where min - the lowest project cost, max - the highest project cost, avg - the average project cost, from all individuals of a given generation

and more diverse. Figure 5 shows how a reduction of the deadline affects the distribution of results in generations. The lower the deadline, the lower the number of bad individuals, which were reproduced to next generations. The percentage of newly created genotypes that passed the *life test* fell from 82 (90% of the deadline set) to 36 (60% of the deadline set). However, it did not influence the results. The average project cost, for different time constraints, is close to the best result, which indicates that most individuals in the population correspond to good quality results. Moreover, the convergence of the algorithm is very similar.

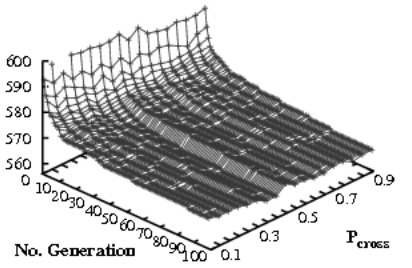
3) Comparison test: Finally, we have performed efficiency test on all 480 project instances where 10 independent runs were computed for each test case. The results were averaged and compared with other methods (Table I). Greedy procedures try to find a resource for each task, in a valid order, according to the smallest increase of the project duration ($Greedy_{time}$) or the project total cost ($Greedy_{cost}$). Another method is a heuristic based on iterative improvements driven by a metric of the gain of optimisation (MAO) [Deniziak, 2004]. It has the capacity of getting out of a local minimum. In [Pawiński and Sapiecha, 2013] the metaheuristic algorithm was adapted to take into account specific features of human resources participating in a project schedule. Their research showed high efficiency of the algorithm for resource allocation. Genetic approaches have similar evolution process



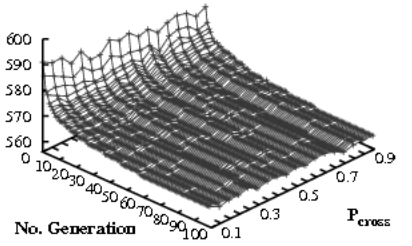
(a) $POP_{size} = 15$, $P_{cross} = 0.8$



(b) $POP_{size} = 30$, $P_{cross} = 0.8$



(c) $POP_{size} = 15$, $P_{mut} = 0.8$



(d) $POP_{size} = 30$, $P_{mut} = 0.8$

Fig. 4. Minimal cost of the project in generations for $Tree_{height} = 10$ and $P_{cross} = 0.8$ (upper row) or $P_{mut} = 0.8$ (lower row)).

TABLE I
THE METHOD COMPARISON RESULTS ($POP = 30, TS_{size} = 3, P_{cross} = 0,8, P_{mut} = 0,8, Tree_{height} = 10$)

Scheduling method	C	T_p	computation time [s]
$Greedy_{time}$	651,89	105,4	70
$Greedy_{cost}$	654,92	106,47	65
MAO	637,94	102,97	4560
GA	619,89	98,57	10062
DGP	599,9	92,69	13650

TABLE II
A COMPARISON OF THE UNCORRECTED SAMPLE STANDARD DEVIATION (S_N)

Scheduling method	S_N
GA	12,83
DGP	4,89

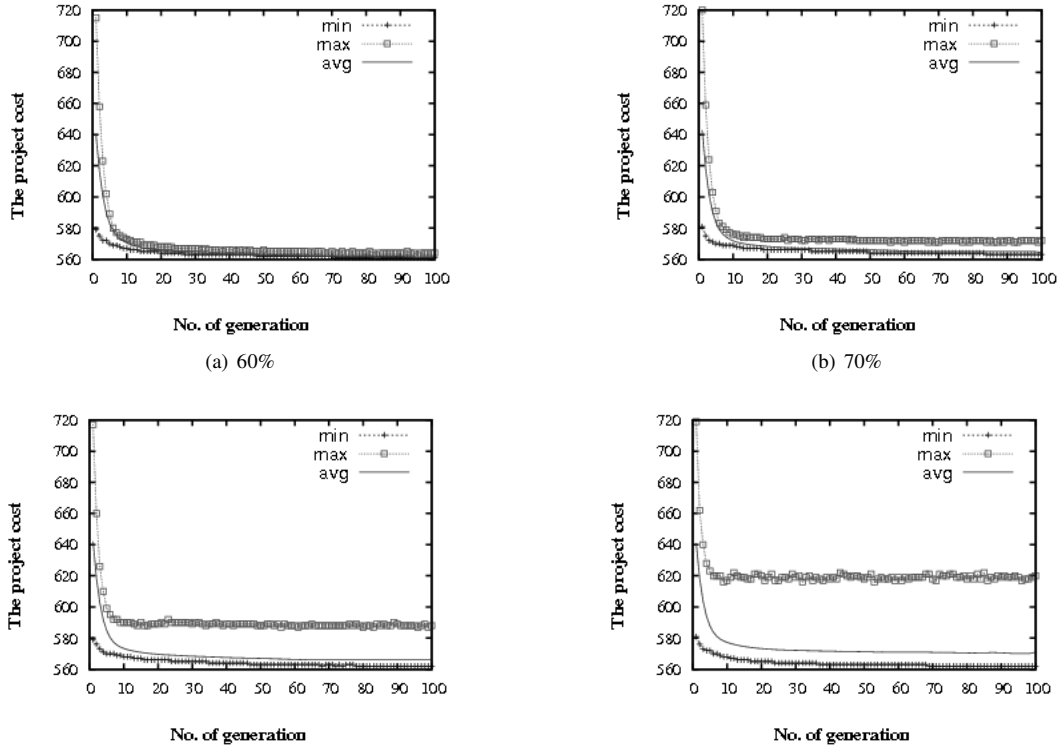
but they differ in a way of coding the genotype. In GA the genotype does not have a tree structure. Genetic operators are performed directly on a sequence of resources corresponding to project activities.

The DGP outperformed the MAO by 5.5% and the greedy methods by 8% as concerns project cost reduction, and by 6% and 12% as concerns project time reduction. Furthermore, the uncorrected sample standard deviation of the DGP was 3-times lower than the deviation of the GA (Table II). However, the DGP was 3-times slower than the MAO and 36% slower than the GA, mainly because of the high number of generations.

VI. CONCLUSIONS

The objective of this research was to introduce and evaluate a new heuristic that can efficiently solve an extension of the RCPSP. It is based on the idea of developmental genetic programming. An algorithm, which was worked out, searches for the best resource allocation strategies in a project. The method of constructing a project schedule takes the form of a decision tree that evolves, instead of evolving the solution itself. The quality of the solution is evaluated after the genotype-to-phenotype mapping.

The fitness function was defined as the project cost. Genetic operators specified for RCPSP were presented as well. To our best knowledge this is the first developmental genetic approach targeting the problem. Three reproduction methods were tested, from which the tournament method ($TS_{size} = 3$) gave the best results. The tournament reproduction ensures that only the best individuals will be selected to the next generation. Then, the influence of various probabilities of mutation and crossover on the algorithm performance was evaluated. Usually, the project cost was lower along with increasing P_{cross} as well as increasing P_{mut}. In the tournament and ranking selection the best results were obtained for $P_{cross} + P_{mut} > 1$. The project cost also decreases as the number of generations increases. Yet, 9 generations is enough



(a) 60% (b) 70%

(c) 80% (d) 90%

Fig. 5. Project cost in each generation for $P_{mut} = 0.8$, $P_{cross} = 0.8$, $POP_{size} = 30$ and with reduced time constraints, where min - the lowest project cost, max - the highest project cost, avg - the average project cost from all individuals of a given generation

to obtain good quality results. Further reduction of the project cost may be achieved by increasing the population size. The influence of the project time constraints on the final results was also tested. The number of bad individuals that were reproduced in subsequent generations was lower when the time constraints were more restricted. But though the genetic operators created 46% less genotypes that passed the *life test*, it did not influence the results. Experimental results showed that DGP is efficient and may be used for solving the extension of the RCPSP. It gives significantly better results than existing methods - it is 5% better than MAO and 8% better than greedy methods.

To our best knowledge this is the first developmental genetic approach targeting the problem. Future work will concentrate on analysing the influence of other parameters of the evolution as well as the influence of different implementations of genetic operators. We will also work on the parallel DGP model to reduce computation time.

REFERENCES

[Ahn and Erenguc, 1998] Ahn, T. and Erenguc, S. (1998). The resource constrained project scheduling problem with multiple crashable modes: A heuristic procedure. *European Journal of Operational Research*, 107(2):250–259. DOI: 10.1016/S0377-2217(97)00331-7

[Alcaraz and Maroto, 2001] Alcaraz, J. and Maroto, C. (2001). A robust genetic algorithm for resource allocation in project scheduling. *Annals of Operations Research*, 102:83-109. DOI: 10.1023/A:1010949931021

[Blazewicz et al., 1983] Blazewicz, J., Lenstra, J. K., and Kan, A. H. G. R. (1983). Scheduling subject to resource constraints: Classification and complexity. *Discrete Applied Mathematics*, 5:11-24.

[Bouleimen and Lecocq, 2003] Bouleimen, K. and Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research*, 149(2):268–281. DOI: 10.1016/S0377-2217(02)00761-0

[Brucker et al., 1998] Brucker, P., Knust, S., Schoo, A., and Thiele, O. (1998). A branch-and-bound algorithm for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 107:272-288. DOI: 10.1016/S0377-2217(97)00335-4

[Deiranlou and Jolai, 2009] Deiranlou, M. and Jolai, F. (2009). A new efficient genetic algorithm for project scheduling under resource constrains. *World Applied Sciences Journal*, 7(8):987-997. DOI: 10.1002/nav.10029

[Demeulemeester and Herroelen, 1997] Demeulemeester, E. L. and Herroelen, W. S. (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43:1485-1492. DOI: 10.1287/mnsc.43.11.1485

[Demeulemeester and Herroelen, 2002] Demeulemeester, E. L. and Herroelen, W. S. (2002). *Project scheduling - A research handbook*. International Series in Operations Research, Management Science, Boston, MA, USA.

[Deniziak, 2004] Deniziak, S. (2004). Cost-efficient synthesis of multiprocessor heterogeneous systems. *Control and Cybernetics*, 33:341-355.

[Deniziak and Górski, 2008] Deniziak, S. and Górski, A. (2008). Kosynteza systemów soc metoda rozwojowego programowania genetycznego. *Wydawnictwo Politechniki Krakowskiej, in polish*, 105(1-I):19-32.

[Deniziak and Wieczorek, 2012] Deniziak, S. and Wieczorek, K. (2012). Evolutionary optimization of decomposition strategies for logical functions. In *Proceedings of 11th International Conference on Artificial Intelligence and Soft Computing*, volume 7269, page 182-189. Lecture Notes in Computer Science. DOI: 10.1007/978-3-642-29353-5_21

[Drexl et al., 2000] Drexl, A., Patterson, J. H., and Salewski, F. (2000). ProGen/px An instance generator for resource-constrained project scheduling

problems with partially renewable resources and further extensions. *European Journal of Operational Research*, 125:59–72. DOI: 10.1016/S0377-2217(99)00205-2

[Frankola et al., 2008] Frankola, T., Golub, M., and Jakobovic, D. (2008). Evolutionary algorithms for the resource constrained scheduling problem. In *Proceedings of 30th International Conference on Information Technology Interfaces*, volume 7269, page 715-722. Information Technology Interfaces.

[Goldberg, 1989] Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co. and Inc., Boston, MA, USA.

[Hartmann, 1998] Hartmann, S. (1998). A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics*, 45:733-750. DOI: 10.1002/(SICI)1520-6750(199810)45:7¡733::AID-NAV5¿3.0.CO;2-C

[Hartmann and Briskorn, 2010] Hartmann, S. and Briskorn, D. (2010). Survey of variants and extensions of the resource-constrained project scheduling problem. *European journal of operational research*, 207:1-15. DOI: 10.1016/j.ejor.2009.11.005

[Jedrzejowicz and Ratajczak-Ropel, 2014] Jedrzejowicz, P. and Ratajczak-Ropel, E. (2014). Reinforcement Learning Strategy for Solving the Resource-Constrained Project Scheduling Problem by a Team of A-Teams. *Intelligent Information and Database Systems*, page 197–206. DOI 10.1007/978-3-642-40495-5_46

[Keller and Banzhaf, 1999] Keller, R. E. and Banzhaf, W. (1999). The evolution of genetic code in genetic programing. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, page 1077-1082. Information Technology Interfaces.

[Kolisch and Hartmann, 2006] Kolisch, R. and Hartmann, S. (2006). Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European journal of operational research*, 174:23-37. DOI: 10.1016/j.ejor.2005.01.065

[Kolish and Sprecher, 1996] Kolish, R. and Sprecher, A. (1996). Psplib - a project scheduling library. *European journal of operational research*, 96:205-216.

[Koza et al., 2003] Koza, J., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J., and Lanza, G. (2003). *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers.

[Koza, 1992] Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA.

[Koza, 2010] Koza, J. R. (2010). Human-competitive results produced by genetic programming. *Genetic Programming and Evolvable Machines*, 11:251-284. DOI 10.1007/s10710-010-9112-3

[Koza and Poli, 2005] Koza, J. R. and Poli, R. (2005). *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*. Springer, New York, NY, USA.

[Mingozzi et al., 1998] Mingozzi, A., Maniezzo, V., Ricciardelli, S., and Bianco, L. (1998). An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management Science*, 44:714-729.

[Möhring et al., 2003] Möhring, R. H., Shulz, A. S., Stork, F., and Utez, M. (2003). Solving project scheduling problems by minimum cut computations. *Management Science*, 49(3):330-350. DOI: 10.1287/mnsc.49.3.330.12737

[Pawiński and Sapiecha, 2013] Pawiński, G. and Sapiecha, K. (2013). Cost-efficient project management based on distributed processing model. In *Proceedings of the 21th International Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2013)*, page 157-163. IEEE Computer Society. DOI: 10.1109/PDP.2013.30

[Pinedo and Chao, 1999] Pinedo, M. and Chao, X. (1999). *Operations Scheduling with applications in Manufacturing*. Irwin/McGraw-Hill, Boston, New York, NY, USA, 2nd edition.

[Skowronski et al., 2013] Skowronski, M. E., Myszkowski, P., Adamski, M. and Kwiatek, P. (2013) Tabu search approach for Multi-Skill Resource-Constrained Project Scheduling Problem. In *Federated Conference on Computer Science and Information Systems (FedCSIS 2013)*, page 153–158. IEEE Computer Society.

[Watson et al., 1992] Watson, J. D., Hopkins, N. H., Roberts, J. W., Steitz, J. A., and Weiner, A. M. (1992). *Molecular Biology of the Gene*. Benjamin Cummings, Menlo Park, CA.

[Zamani, 2013] Zamani, R. (2013). Integrating iterative crossover capability in orthogonal neighborhoods for scheduling resource-constrained projects. *Evolutionary Computation*, 21(2):341–360. DOI: i:10.1162/EVCO_a_00085