Position Papers of the 20th Conference on Computer Science and Intelligence Systems (FedCSIS) pp. 37–40
DOI: 10.15439/2025F5892
ISSN 2300-5963 ACSIS, Vol. 44

Constructive genetic algorithm with penalty function for a concurrent real-time optimization in embedded system design process

Adam M. Górski
0000-0003-3821-5333
Jagiellonian University
Faculty of Physics, Astronomy
and Applied Computer Science
ul. Prof. Stanisława
Łojasiewicza 11,
30-348 Kraków, Poland
Email: a.gorski@uj.edu.pl

Maciej J. Ogorzałek,
fellow IEEE
0000-0003-3314-269X
Jagiellonian University
Faculty of Physics, Astronomy
and Applied Computer Science
ul. Prof. Stanisława
Łojasiewicza 11,
30-348 Kraków, Poland
Email: maciej.ogorzalek@uj.edu.pl.

***Abstract*—In this paper we present a genetic programming based constructive algorithm with penalty function for a concurrent real-time optimization in embedded system design process. Proposed approach uses genetic programming mechanism to optimize detecting and assignment of unexpected tasks process in embedded system design. Unlike others methodologies the approach described in this paper uses a penalty in objective function in optimization process. As a result during the evolution generations of individuals can also contain solutions which violate time constraints. Thus the approach is more proof to stop in local minima of optimizing parameters. Therefore the final result could be better adapted to the environment and the optimization process can be cheaper and more effective.**

***Index Terms*—Genetic Programming, Concurrent Real-Time Optimization, Embedded Systems, Artificial Intelligence.**

I. Introduction

EMBEDDED system design process [1] can be split on four phases[2]: modeling, implementation, validation and assignment of unexpected tasks. Unexpected tasks [2][3] can appear when the architecture of embedded system is produced, all known tasks are assignment to available resources and the system works in a target environment. In [4] authors proposed a methodology for assignment of unexpected tasks for a group of embedded systems. Unexpected tasks that appeared were the result of cooperation of the systems in bigger environment. The first methodologies [2][5] proposed for assignment of unexpected tasks have one major weakness – unexpected tasks needed to be detected externally. All values of time and cost of execution needed to be given for every task. In [6] the authors proposed an algorithm which was able to detect unexpected tasks and assign them to appropriate Processing Element (PE). The authors indicated that some of unexpected situations can be solved as a result of connection of some number of subtasks of known tasks. However not only one connection of subtask leads to solve unexpected situation. On the other side not every connection give the solution. Connection of a subtasks that gives an appropriate solution needs to be assignment on one of available resources. The problem is to find which connection of subtasks is better. Such a problem was called picking an apple problem. Generally the optimization process can be split into two phases. Each phase impacts another in a real-time. That is why this type of optimization was named concurrent real-time optimization. Further information about the problem and are given in next section. In [7] the authors proposed the solution of such a problem in IoT design. In [6] genetic algorithm was proposed to solve the problem in embedded system design. Genetic programming methodology [8] was also presented for such a problem. The biggest disadvantage of the methodology was a constructive nature of the algorithm. Such group of methodologies [9] [10] have low complexity but are prone to stop in local minima of optimizing parameters. It is caused because such methodologies construct the system by making decisions step by step for every task separately. Iterative improvement algorithms [11] [12] start from suboptimal solutions, usually the fastest, and by local changes try to improve the system quality. Such algorithms can escape local minima however the results are still suboptimal. In [13] the authors provided the genetic programming based iterative improvement method for the problem. However the biggest disadvantages of the methodology was that only valid individuals could be investigated in the evolution process. Therefore some of the solutions could be unobtainable. Concurrent real-time optimization occurs not only in hardware design. The solution for such kind of optimization was also proposed in game theory [14]. Proposed methodology belonged to metaheuristics group. The authors

·The publication has been supported by a grant from the Faculty (Faculty of Physics, Astronomy and Applied Computer Science) under the Strategic Programme Excellence Initiative at Jagiellonian University.

proposed a grey wolf optimizer to find an automatic solution of computer games.

In this paper we propose a genetic algorithm based methodology [6][16] with penalty function for concurrent real-time optimization in embedded system design process. Unlike other approaches we investigate in evolution process not only valid solutions. Therefore the algorithm is more able to escape local minima of optimizing parameters.

The paper is organized as follows: next section are preliminaries, then the algorithm is described. The fourth section contains experimental results. At the and the conclusions and directions of future work are presented.

II. Preliminaries

A. Embedded systems

Embedded systems are computer systems mostly microprocessor or microcontroller based. They were created to execute some special group of tasks. most of modern systems are solved as distributed once. Such kind of systems are consisted of two kinds of resources: processing elements (PEs), responsible for executing the tasks, and communication links (CLs) responsible for providing communication between PEs. There are two basic kinds of PEs: programmable processors (PPs) and hardware cores (HCs). PPs are universal resources able to execute more than one task. HCs are specialized resources dedicated to execute only one task. Therefore unexpected tasks can be executed only by PPs. The behaviour of the system is specified by an acyclic directed graph called an extended task graph G = (V, E). Each node $v_i \in V$ in the graph is a task, each edge $e_{ij} \in E$ describes the amount of data transferred between two connected tasks. The transmission time t_{ij} is equal to:

$$t_{ij} = \frac{e_{ij}}{b} \quad (1)$$

where b is a bandwidth of a communication link. Fig. 1 below presents the example of a task graph.

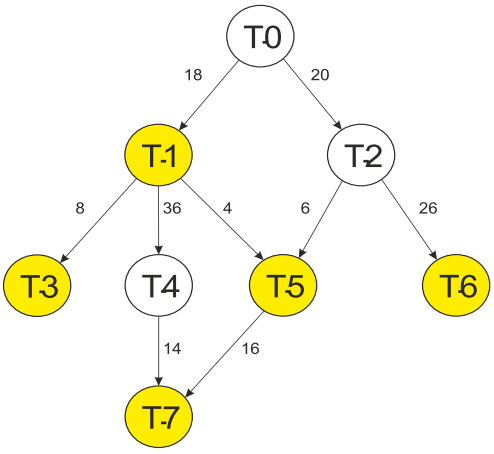

Fig 1. Example of an extended task graph

The graph contains eight tasks. The nodes with yellow color (T1, T3, T5, T5, T7) marks the tasks that can be split on subtasks. The overall cost of a system C_o is described by the following formula:

$$C_o = C_a + \sum_{i=1}^{n} C_i + k * (t - t_{max}) \quad (2)$$

where n is a number of tasks in an extended task graph, t_{max} is a time constrain, k is a parameter given by the designer which decides about the penalty function and therefore how is the weight of violation of time constraints. The unit of k is [c/t] where c is a unit of cost and t is a unit of time. The goal of the optimization is to find the solution with the lowest value of C_o.

B. Concurrent real-time optimization – picking an apple problem

It is possible to pick up an apple on many ways. Each of them demands different parameters to optimize and different tasks to execute. The question is how to find out which way is better without picking up the apple. The problem can be split into two phases. The first phase is responsible for the choice of optimizing parameters. The second phase makes the optimization and verifies the choice made in the first phase. During the process it can be found out that some of the ways of solving the problem do not lead to success. It is not always known at the beginning. In such a case after executing some of the tasks and starting optimization process the tasks to execute and optimization parameters are changed. Therefore each of the phases can impact another in real time. The change of one phase changes another. Some unexpected situations can also happen which demands to execute some unexpected tasks. The most important issue is how to compare the results if every can demand different parameters to optimize or optimizing parameters can change during the process. Every solution can be characterized by global common parameters which can be used to compare the results. Such parameters can be for example cost of the solution or time of execution of all the tasks.

Such a problem occurs in embedded system design. If system meets unexpected situation it can be solved on many ways. Each way demands different tasks to execute. The problem is to find the optimal way to execute unexpected tasks and to find the appropriate hardware components to execute them. The solutions can be compared using two global parameters: time and cost of execution of all the tasks. As it can be easily observed, if the time is getting lower the cost is rising. Such relation is not proportional. Therefore, designing of embedded systems belongs to pareto group of problems [15].

III. The Algorithm

Unexpected tasks can appear in every moment of system life, after every task. After appearing of such a task it needs to be inserted on extended task graph as a separate task. Then all the tasks need to be split on possible number of subtasks. The algorithm starts with generating the initial population. The number of individuals in population is dependent on

a number of programmable processors p and a number of tasks n in the extended task graph. It is equal to:

$$\Pi = \alpha * p * n \quad (3)$$

where α is given by a designer. It controls the size of the population. In each of the individuals unexpected tasks are solved as a random connection of randomly chosen number of subtasks. Not every connection gives the solution of unexpected situation. Such solutions are not passed over. Next generations of individuals are created using standard genetic operators: crossover, mutation, cloning and selection. In this paper we decided to choose rank selection. After generating each population the genotypes are ranked by cost. All of the individuals on a rank list have probability of being chosen during the evolution process. The probability P depends on a position r of an individual in a rank list. It is described by the following equation:

$$P = \frac{\Pi - r}{\Pi} \quad (4)$$

Crossover selects Ψ individuals and randomly connects them in pairs. Then for each genotype in each pair randomly a cutting point is chosen. The genes of two parents are swapped. The number of individuals created by crossover is presented on equation 4 below:

$$\Psi = \gamma * \Pi \quad (5)$$

where γ is a parameter given by the designer, γ ϵ (0,1).

Mutation selects Ω genotypes. Then randomly a gene is chosen. The number of a PE in the gene is substituted by another. The mutation can also change the connection of subtasks solving unexpected situation. The number of individuals created using mutation operator is equal to:

$$\Omega = \beta * \Pi \quad (6)$$

where β ϵ (0,1) and is given by a designer.

Cloning copies Φ individuals to a new population without any changes. Φ is equal to:

$$\Phi = \delta * \Pi \quad (7)$$

where δ ϵ (0,1). It is given by a designer.

To have the same number of individuals in all of the populations the sum of the parameters β, γ and δ needs to be equal to 1:

$$\beta + \gamma + \delta = 1 \quad (8)$$

The algorithm finishes its execution after ε generations without a better result.

IV. Experimental results

In this section the results of the experiments are presented. The results were compared with genetic programming methodology [13] proposed by Górski and Ogorzałek (GP 2025). Table 1 contains the results. The results were made for benchmarks with 10, 20 and 30 nodes. The parameters were set as follows for both of algorithms: α = 100, γ = 0,7, β = 0,2, δ = 0,1 and ε = 5. The first results seem promising. However it is needed to underline that presented results are first obtained and the algorithm needs further investigation with different parameters, time constrains and bigger graphs.

Algorithm presented in this paper (GA 2025) was able to provide better results for every benchmarks. For the graph with ten nodes the difference between the best results obtained by GP 2025 and GA 2025 was the lowest – it was equal only 15 units of cost. The cost of the best solutions generated by both algorithms was the same and equal to 100. That could be an effect of a small size of the graph. For a such a graph the search space is smaller and maybe it could be reasonable to decrease the value of α parameter. The difference between obtained values of cost for a graph with 20 nodes was the greatest – more than 700 cost units (1643 for GA 2025 and 2358 for GP 2025). Such a difference is surprising however we cannot forgot about probabilistic nature of the algorithms. As a consequence of such a type of algorithms one or a few results can be very different from the majority. For a graph with 30 nodes there was not such a big difference of costs – it was appropriately 1643 for GA 2025 and 2358 for GP 2025. It is worth to mention that even that GP 2025 produced the results which were more expansive the time of the solutions was faster in most of cases than the time of results generated by GA 2025. Such a situation was expected because, as it was mentioned before, investigated problem in hardware design belongs to pareto group of problems. It also can be observed that GA 2025 generated less populations than GP 2025 – 17, 14 and 18 for graphs with 10, 20 and 30 nodes meanwhile GP 2025 produced appropriately 22, 15 and 23 generations.
graph.

Table I.
Experimental results

Graph	GA 2025			GP 2025		
	cost	time	generation	cost	time	generation
10	197	100	17	212	100	22
20	1643	2497	14	2358	2394	15
30	1997	2956	18	2244	2873	23

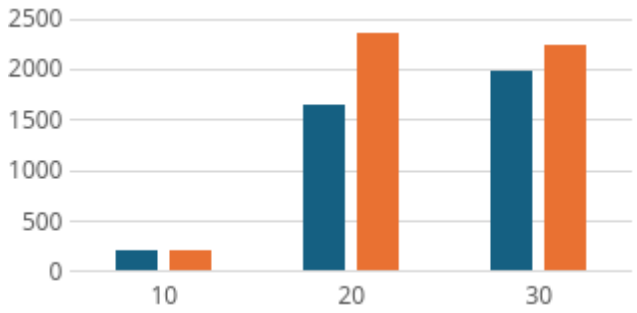

Fig 2. Comparison of obtained results

In fig. 2 the graphical comparison of the results were presented. It contains best obtained results for every used benchmark. Algorithm GP 2025 was compared with GP 2024 [8] and was more efficient.

V. Conclusions

In this paper a constructive genetic algorithm for a concurrent real-time optimization in embedded systems design process was proposed. Solving the problem in hardware design can make the design faster and cheaper. It can also help with adapting embedded systems to a changing environment and thus making the systems more universal.

In the paper only first results were presented. Therefore the algorithm needs more examination. The experiments should be made using bigger graphs, different parameters and time constrains.

In the future we plan to deliver more algorithms to solve the problem investigated in hardware design. It seems that good direction is to develop genetic programming solutions. We also plan to propose new solutions to concurrent real-time optimization problem in other areas too, not only in embedded system design. Therefore the future work will be divided into two directions. The first direction will include improvement of proposed algorithms for hardware design. The second direction will be concentrated on investigated problem – its constrains, areas of appearance and searching its different solutions.

Acknowledgment

The publication has been supported by a grant “Solving real-time optimization problems” from the Faculty of Physics, Astronomy and Applied Computer Science under the Strategic Programme Excellence Initiative at Jagiellonian University.

References

[1] G. C. Duarte, D. S. Loubach and I. Sander, "High-Level Reconfigurable Embedded System Design Based on Heterogeneous Models of Computation," in *IEEE Access*, vol. 13, 2025, pp. 63918-63934,

[2] A. Górski and M., J. Ogorzałek, “Assignment of unexpected tasks in embedded system design process”. *Microprocessors and Microsystems*, Elsevier vol. 44, 2016 pp. 17– 21.

[3] A. Górski and M., J. Ogorzałek, “Auto-detection and assignment of unexpected tasks in embedded systems design process*”. Proceedings of the 23rd International Workshop of the European Group for Intelligent Computing in Engineering*, 2016, pages 179 – 188.

[4] A. Górski and M., J. Ogorzałek, “Assignment of unexpected tasks for a group of embedded systems. *IFAC-PapersOnLine*, vol. 51, Issue 6, 2018, pp. 102-106.

[5] A. Górski and M., J. Ogorzałek, “Assignment of unexpected tasks in embedded system design process using genetic programming”. *Proceedings of the 6th International Conference on the Dynamics of Information Systems (DIS 2023)*, Lecture Notes in Computer Science, vol. 14321, Springer, Cham., 2024, pp 93 – 101.

[6] A. Górski and M., J. Ogorzałek, “Concurrent real-time optimization in embedded system design process using genetic algorithm”. *Progress in Polish Artificial Intelligence Research 5: proceedings of the 5th Polish Conference on Artificial Intelligence* (PP-RAI’2024), 2024, pp. 331-337.

[7] A. Górski and M., J. Ogorzałek, “Concurrent Real-Time optimization of detecting unexpected tasks in IoT design process using GA”. *Late Breaking Papers from the IEEE 2023 Congress on Evolutionary Computation*, Chicago, IL, USA, IEEE, 2023, pp. 74 – 77.

[8] A. Górski and M., J. Ogorzałek, “Detecting and assignment of unexpected tasks in SoC design process using genetic programming”. *Proceedings of the 21st International SoC Design Conference (ISOCC),* Sapporo, Japan, august 2024 pp. 398-399.

[9] B. P. Dave, G. Lakshminarayana and N. K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 7, no. 1, March 1999, pp. 92-104.

[10] S. Q. Liu and E. Kozan, “A hybrid metaheuristic algorithm to optimise a real-world robotic cell”, *Computers & Operations Research.* Vol. 84, Elsevier, 2017, pp. 188-194.

[11] J. A. Austin, M. A. Barras, and C. M. Sullivan. "Safe and effective digital anticoagulation: a continuous iterative improvement approach." *ACI open* 5.02 (2021), pp. e116-e124.

[12] H. Lu, X. Zhang, S. Yang, A, “earning-based iterative method for solving vehicle routing problems” *International Conference on learning Representations*, 2020.

[13] A. M. Górski and M., J. Ogorzałek, “Genetic programming iterative improvement algorithm for a concurrent real-time optimization in embedded system design process” *Proceedings of the 6th Polish Conference on Artificial Intelligence* (PP-RAI’2025), 2025 (in press).

[14] A. M. Górski and M., J. Ogorzałek, “Grey wolf optimization algorithm for a concurrent real-time optimization problem in game theory” *Proc. Journal of Automation, Mobile Robotics and Intelligent Systems*, vol. 19 no. 2, 2025, pp. 65-72.

[15] S. Mahajan, A. Chauhan, and S. K. Gupta “On Pareto optimality using novel goal programming approach for fully intuitionistic fuzzy multi-objective quadratic problems”. *Expert Systems with Applications*, vol. 243, Elsevier, 2024.

[16] K. Gmyrek, M. Antkiewicz and P. Myszkowski ” Genetic Algorithm for Planning and Scheduling Problem -- StarCraft II Build Order case study*” Proceedings of the 18th Conference on Computer Science and Intelligence Systems, Annals of Computer Science and Information Systems*, vol. 35, 0223 pp. 131–140.