sciendo

VOLUME 19, N° 2 2025
Journal of Automation, Mobile Robotics and Intelligent Systems

Grey Wolf Optimization Algorithm for a Concurrent Real-time Optimization Problem in Game Theory

Submitted: 12th November 2024; accepted: 13th January 2025

Adam M. Górski, Maciej Ogorzałek

DOI: 10.14313/jamris-2025-016

Abstract:
This paper presents a grey wolf algorithm for a concurrent real-time optimization problem in searching for an optimal game-solving solution. There are many solutions to the game. Each solution can demand different optimal values of different parameters. However, some ways the players try to solve the game do not lead to success. The optimization problem consists of two phases. Each phase impacts the second one in real time. The first phase is responsible for the optimization of the parameters. The second phase validates the choice and optimizes the parameters. As an optimization method, we chose grey wolf optimization. At the beginning, the algorithm generates several solutions. The solution with the value of the parameters closest to maximum is the position of an alpha wolf. The rest of the solutions are, according to the values of the parameters, split into the positions of beta, delta, and omega wolves.

Keywords: *concurrent real-time optimization, grey wolf optimization, metaheuristics, swarm intelligence, game theory*

1. Introduction

Games can very realistically simulate a lot of real situations of daily life. Such simulations are easy to make, cheap, and have no consequences in real life. They can also improve existing solutions or indicate their weak points. Instead of putting human life in danger or losing some expensive hardware, it is better to play a game. Modern computer games have become more complex. They allow for checking more and more aspects of the situation they simulate. However, checking every possible solution to solve the game can take too much time and is too expensive. What is more, some ways do not lead to success. Therefore, it is very important to find the best solutions and eliminate the wrong ones. But what does the best solution mean? Games can be solved in many ways. Each way demands different values of many different parameters. The question is which parameters the players should choose and what their optimal values will be when the time to solve the game is limited. As can be observed, the optimization process might be split into two phases. The first phase chooses the parameters that need to be optimized. The second phase verifies the choice and optimizes the parameters.

Every change in the second phase makes changes in the first one. If the first one is modified, the optimization in the second one also must be changed. Therefore, the process describes concurrent real-time optimization [1]. Such a type of optimization was proposed by Górski and Ogorzałek to find unexpected tasks in the IoT design process and their optimal assignment.

In this paper, a grey wolf algorithm was implemented for a concurrent real-time optimization process of searching for the best solution to solve computer games. To the best of our knowledge, such an optimization problem has not been investigated in game theory so far. Solving this problem can help to automatically find the solution to computer games.

The paper is organized as follows: the second chapter includes related work, the third one describes the implementation of grey wolf optimization to investigated problem. In the next section the experimental results are given. The last chapter contains conclusions and directions of future work.

2. Related Work

Concurrent real-time optimization [1] is a new kind of optimization. It was proposed by Górski and Ogorzałek [1]. Such a type of problem indicates that there are two optimizing phases. Each phase impacts another in real time. Changes in one phase make changes in the second one in real-time. The first phase is responsible for the choice of parameters to optimize. The second phase validates the choice and optimizes it. However, during the optimization process, the parameters may be changed. Such a problem was called a picking an apple problem. There are many possibilities to pick an apple. For example: climbing a tree, shaking a tree, using a tool or a ladder, etc. Depending on the choice, there are different parameters to optimize. It is very hard to establish which way is best. Moreover, in some cases, not all the ways to pick up an apple are possible. The solutions can be characterized by some common parameters that can be used to evaluate the quality of the results and choose the best one. The same situation can be found in game theory. There are many possible ways to solve a game. Each one demands different actions taken by a player and therefore different values of available parameters.

However, depending on the situation in a gameplay, increasing the values of some of the parameters does not lead to solving the game. The common parameter for every solution is the time necessary to solve the game. Such a problem was solved in IoT [1] and embedded systems [2] design to detect and assign unexpected tasks. In [33], the authors proposed a genetic programming approach for detecting and assigning unexpected tasks in the SoC design process. So far, every one of the algorithms solving concurrent real-time optimization problems has been proposed in hardware design. However, the problem in hardware design is different. The hardware was specified by the task graph [34]. Therefore, the hardware is needed to execute some tasks. The architecture consisted of two kinds of processing elements: programmable processors (PPs) and hardware cores (HCs). PPs were able to execute more than one task, while HCs were dedicated to executing only one task. The tasks were characterized by two parameters. Usually, it was time and cost. The faster the architecture, the more expensive it was. Therefore, in hardware design, concurrent real-time optimization belongs to the Pareto group of problems [35]. In game theory, such a situation does not occur. Modifying one feature of the character does not decrease the value of another. In [36], Górski proposed an extension of a task graph for real-life problems. Such an extended task graph can allow for the proposal of solutions in more areas.

2.1. Game Theory

Game theory describes the way to solve a game using mathematical formulas. Every game consists of four parts [3]:

- Players – persons who play the game by controlling characters.
- Actions – actions that are taken by the players to solve the game.
- Strategies – a way chosen by the players to solve the game.
- Payoff – return (positive or negative) obtained by the players after taking the actions.

The characters controlled by the players during the gameplay can be characterized by many, sometimes different, parameters (abilities). Generally, the values of those parameters are not constant during gameplay. The opponents, often controlled by the computer, also have different values of the same parameters. The values of the parameters for the characters controlled by the computer can also be modified. In [4], the authors proposed using graph databases to improve artificial intelligence in games. The players must choose the strategy for evolving the characters and solving the game. In multiplayer games, the strategies must include an influence of one player's behavior on the rest [5]. In [6], the authors proposed a graph-based model to automate game story generation. Unfortunately, the proposed solution did not include every aspect of the gameplay.

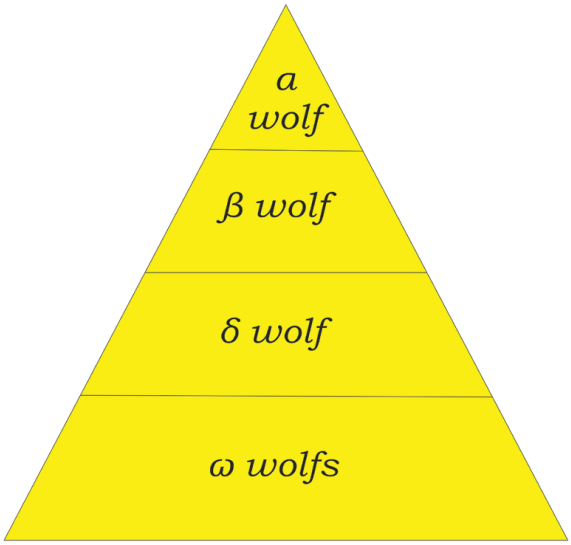

Figure 1. Chierarchy in a group of wolves

In game theory large group of games are serious games [7]. The purpose of such games is not only entertainment. They are used in nurse education [8], language learning [9], learning cultural heritage [10], on environmental management [11] and in many other areas of daily life mostly common with training and education.

2.2. Grey Wolf Optimization

Grey wolf optimization (GWO) was proposed by Mirjalili, Mirjalili, and Lewis in 2014 [12]. It is a kind of swarm intelligence algorithm, like particle swarm optimization [13], grasshopper optimization algorithm [14], whale optimization algorithm [15], or Dragonfly Algorithm [16]. GWO is a simulation of hunting prey by a group of wolves. In every group of wolves, there is a leader called the alpha wolf. The leader is closest to the prey. The next position belongs to Beta Wolf. A beta wolf is a candidate for becoming an alpha. The third in order is the delta wolf. The last in a hierarchy are omega wolves – the rest of the group. Such a hierarchy is presented in Figure 1 below.

Hu Pan and Chu proposed a binary grey wolf optimizer (BGWO) to solve binary problems [17].

GWO is well known to stop at the local minima when optimizing parameters. Therefore, during the last few years, some modifications have been proposed. One of the modifications is based on the usage of maps like circle maps [18] and chaotic maps [19, 20]. Another improvement to GWO is an adaptive grey wolf optimizer (AGWO) [21]. Such a modification allows the algorithm to provide optimal values faster by using a three-point history of parameter values. In [22], the authors proposed using chaos theory to improve the algorithm by making it more prone to avoiding local minima when optimizing parameters. Hybrid solutions were also proposed, like connections of GWO and β-hill climbing algorithm [23] called β-hill climbing grey wolf optimizer (β-HCGWO).

In [24], a beetle antenna strategy that allows the α wolf to hear and thus improve the hunting strategy. Yang and others proposed to modify the position update strategy by adding information about ω wolves, making the iterations nonlinear [25]. In [26], the authors proposed a group-state competition strategy to extend the search space of GWO. Nadimi-Shahraki, Taghian, and Mirjalili proposed an improved grey wolf optimizer (I-GWO) [27]. I-GWO implemented a dimensional learning hunting strategy. The strategy was based on the behavior of a single wolf hunting strategy in nature.

Grey wolf optimization was implemented to: parameter identification of photovoltaic cells [28], task scheduling [29, 30], solving engineering problems [27], path planning [31,32], and many others.

3. The Methodology

Every game can be solved in many ways. Every way demands a different combination of values of different parameters. In most games, there is a main character controlled by a player. The character has a set of parameters. In our methodology, such a set is represented by a vector v consisting of *n* numbers. The vector is presented below:

$$v = [v_1, v_2, \ldots v_n] \tag{1}$$

The player can evolve the character during the game. However, in many cases, time for evolution can be limited. Moreover, the opponents can evolve controlled characters during the gameplay, too. Special features of the opponents can also make evolving one of the parameters useless. To solve the game, one of the winning combinations of all the parameters must be found. Achieving the set of parameters that allows solving the game can take different times. Therefore, the time of evolution, the character, or the number of iterations of evolution are common parameters that allow us to evaluate the quality of the results. In this paper, we concentrate on finding the best way to solve the game, understood as the minimum number of iterations (I_b) demanded to achieve the appropriate set of parameters:

$$I_b = \min(I) \tag{2}$$

Where I is the number of iterations.

At the beginning, a population of m wolves is randomly created.

$$m = n_v * n_p \tag{3}$$

Where: n_p is the number of optimizing parameters and nv is the number of winning combinations of the parameters.

Next, all solutions are divided into appropriate groups: alpha, beta, delta, and omega. The distance (D) between a grey wolf and its prey can be calculated using the equation below:

$$D = |X_p * A - X_w| \tag{4}$$

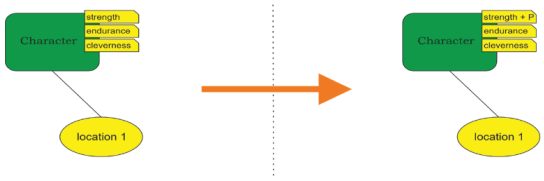

Figure 2. A sheaf graph presenting a single iteration

In Equation (4), X_p describes the coordinates of a prey, and X_w describes the coordinates of the grey wolf. A can be calculated as follows:

$$A = 2 * d \tag{5}$$

Where d is a random value between 0 and 1, as was said before, in the game, there is an established set of vectors that includes minimum combinations of values that allow defeating the opponent. Those are potential positions that allow the wolves to attack the prey. Therefore, the problem is multicriteria. The value of every parameter cannot be decreased. For example, the character cannot lose points of strength after training. The proposed algorithm chooses randomly one of the parameters, v_c, and changes its value in the following way:

$$V = v_c + |P| \tag{6}$$

Where P can be established using the following equation:

$$P = 2 * d * r - r \tag{7}$$

Where r is changed proportionally to the number of iterations from 2 to a maximum of 0 in each iteration, but this value is calculated for each parameter separately. This operation shortens the distance between the wolves and the prey. Such a situation is presented on a sheaf graph [6] below:

Figure 2 presents a simple production (one iteration) – increasing the strength of a character. The character is in location 1 and is characterized by 3 parameters: strength, endurance, and cleverness.

The number of iterations is limited. Because not every way leads to winning a game, the obtained result can be invalid in some runs of the algorithm. Therefore, using the methodology presented in this paper can help to establish the sets of movements that will not allow one to win the game. The goal of the algorithm is to find the fastest way to solve a game. As the fastest way to solve a game, we understand the minimal number of iterations necessary to win a game.

4. Experimental Results

To check the quality of the obtained results, we decided to analyze a scenario of a part of an arcade game. In the scenario, the player chooses one of the characters belonging to three groups: magicians, knights, and villagers.

Every character has some abilities (parameters), such as strength, endurance, magic ability, etc. The player evolves the chosen character in a possible number of iterations (I_{max}). In each iteration, it is possible to evolve only one ability. There are a few combinations of the values of parameters that allow one to win the game and defeat the opponent.

As was written before, to the best of our knowledge, this paper is the first to deal with a concurrent real-time optimization problem in game theory. Therefore, it is very hard to compare the results because of the lack of solutions for such a defined problem. We decided to compare the obtained results with a random algorithm. Such an algorithm generates the initial solution randomly and randomly selects one of the parameters and increases it by a randomly chosen value from 0,1 to 1. In the future, we plan to propose more approaches, like, for example, PSO, for such a problem and compare the results, but so far, other algorithms for concurrent real-time optimization in game theory have not been developed. As was said before, existing algorithms solving concurrent real-time optimization in hardware design cannot be used in game theory because of the specification of the environment in which they work. They work with different constraints, different numbers of optimizing parameters, different behavior of the parameters (Pareto problem), and other environmental conditions. It does not mean that it is impossible to use genetic programming or genetic algorithms to solve the problem in game theory, but such algorithms, which will be equal to work in the game environment, need to be developed.

The experiments are divided into two groups. The first group of experiments was made for three evolvable parameters. The second group of experiments was made for four evolvable parameters. In Table 1 below, the obtained results for three abilities of a character (parameters) are presented. The experiments were made for different numbers of winning combinations (for five winning combinations). A number of winning combinations in Tables 1 and 2 were marked with the letter w. For each number of winning combinations, 50 tries were made. We believe that such a number of runs is enough to provide a good discussion of the results. In Table 1, the results obtained depend on the number of maximum iterations. I_{max} is a constraint on the number of iterations. I_b represents the minimum iteration in 50 runs in which the wolves achieved the position allowing them to attack the prey. I_a is the average number of iterations from all of the valid runs. All the experiments were made for I_{max} equal to: 5, 10, 20, and 50.

In the first set of experiments, there were four winning combinations of three possible parameters. As expected, one of the target vectors was obtained in the lowest average iteration when the constraint was the sharpest (5 iterations). The value was equal to 4.

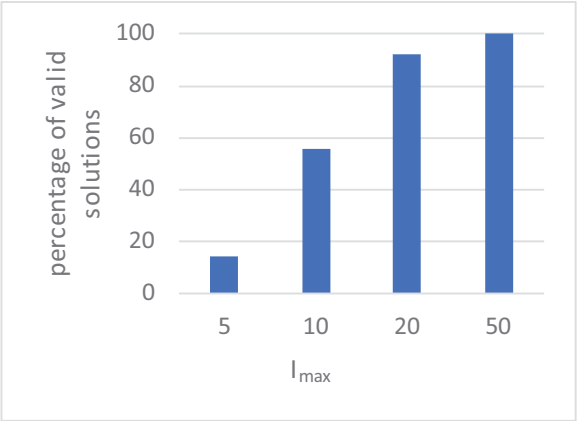

Figure 3. A percentage presentation of a number of valid results for the first set of parameters for the first group of experiments

The average number of iterations in which the winning combination of parameters was achieved grew with the number of possible iterations – 7 for I_{max} equal to 10, 10 for I_{max} equal to 20, and 11 for I_{max} equal to 50. The percentage value of valid runs (the try when a valid result was obtained) was the lowest for the sharpest constraint. When the maximum number of possible iterations was equal to 5, valid solutions were obtained only in 14% of runs. When the number of possible iterations was equal to 10, the number of valid runs increased to 56%. For I_{max} equal to 20, the percentage value of valid obtained results was equal to 92%. For $I_{max} = 50$ in every run valid solution was generated. As can be easily observed, the results obtained by the grey-wolf algorithm were much better than those obtained by the random solution. The best obtained results using the random algorithm were: 9 (for I_{max} equal to 10 and 50) and 10 (for $I_{max} = 20$). The percentage of valid solutions is also worse for the random algorithm. Meanwhile for $I_{max} = 50$ and 20 the results were similar (respectively 100 and 88), for a lower value of I_{max} operator ($I_{max} = 10$) it was only 8%. For $I_{max} = 5$, the random algorithm was not able to generate any valid solution. Figure 3 shows a graphical presentation of the generated valid solution depending on the I_{max} constraint for results obtained by the Grey-wolf algorithm.

For the second set of experiments (five winning combinations and three parameters), the statistics were very similar to those of the first set. However, as we expected, the percentage of achieved winning solutions grew. Such values were achieved because when the number of possible solutions was greater, it should be easier to obtain a valid solution – there is one more winning combination of the parameters. The results were equal to 20% for $I_{max} = 5$, 60% for $I_{max} = 10$, 94% for $I_{max} = 20$, and 100% when I_{max} is equal to 50. That dependency was presented in Figure 4. The percentage of winning solutions for a random algorithm was also higher. For an Imax equal to 20 and 50, it was 92 and 100 percent. For $I_{max} = 10$, it was 18. For $I_{max} = 5$, the algorithm was not able to generate a valid solution.

Table 1. The results for three parameters

I_{max}	GW 2024					Random		
	m	w	I_b	I_a	% of winning solutions	I_b	I_a	% of winning solutions
5	12	4	**3**	**4**	**14**	–	–	0
10	12	4	**4**	**7**	**56**	9	10	8
20	12	4	**4**	**10**	**92**	10	16	88
50	12	4	**4**	**11**	**100**	9	16	100
5	15	5	**4**	**5**	**20**	–	–	0
10	15	5	**4**	**8**	**60**	8	10	18
20	15	5	**4**	**11**	**94**	9	16	92
50	15	5	**5**	**12**	**100**	9	16	100

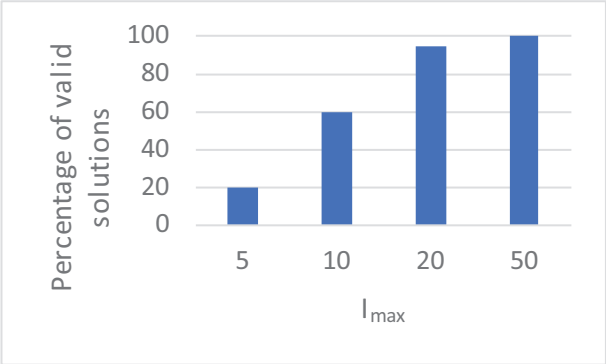

Figure 4. A percentage presentation of a number of valid results for the second set of parameters for the first group of experiments

The experiments were made in the same way as for the first group – with different number of possible iterations: 5, 10, 20 and 50. After comparing the results with the first group it can be noticed that the best results for each I_{max} were generated later.

For the first set (four winning solutions), when $I_{max} = 5$ and $I_{max} = 10$, it was in the 4th iteration, for $I_{max} = 20$ in the 6th iteration, and in the 5th iteration for $I_{max} = 50$. In the second set, the results were as follows: 5th iteration for Imax equal to 5, 10, and 20, and 6th iteration for $I_{max} = 50$. The average number of iterations has also grown. For the first set of experiments, it was equal to: 5 ($I_{max} = 5$), 9 ($I_{max} = 9$), 15 ($I_{max} = 20$), and 19 ($I_{max} = 50$). In the second part of the second group of experiments, the average results were equal to: 5 for $I_{max} = 5$, 9 for $I_{max} = 10$, 14 for $I_{max} = 20$, and 16 for $I_{max} = 50$. Such results of experiments were expected because in the second group of experiments, there were more parameters that needed to be evolved by a player, thus generating a valid combination of parameters could take more time. Similarly, as for the first group of experiments, the average value of iterations decreased when there were more target combinations of the parameters that allowed solving the game. In Figure 5, the dependency of the percentage of generating valid solutions for different maximum numbers of possible iterations for a grey wolf algorithm was presented.

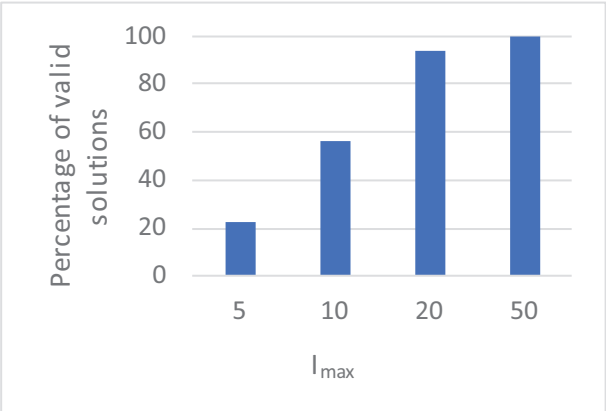

Figure 5. A percentage presentation of a number of valid results for the first set of parameters for the second group of experiments

The lowest value of valid solutions, 22%, was obtained when $I_{max} = 5$. For $I_{max} = 50$ in every run of the algorithm, a valid solution was generated. For $I_{max} = 10$, it was 56%, and for $I_{max} = 20$ it was 94%. A random algorithm was not able to provide a valid solution for Imax equal to 5 and 10 for both parts of the second group of experiments. Even for $I_{max} = 20$, the random algorithm provided only 10% (for four winning solutions) and 4% (for five winning solutions) of valid results.

The algorithm was also slower than the grey-wolf algorithm – it needed to generate, on average, for $I_{max} = 20$: 19 (w = 4) and 20 (w = 5) iterations and for $I_{max} = 50$: 27 (w = 4) and 25 (w = 5) iterations. The graphical representation of the percentage of generating valid solutions for different numbers of Imax for the second part of the experiments for a grey wolf algorithm was presented in Figure 6 below.

Like for the first group of experiments with increasing the number of winning combinations of the parameters, the percentage of obtained valid solutions was growing. It was equal to 30% for $I_{max} = 5$, 68% for $I_{max} = 10$, 94% for $I_{max} = 20$, and 100% for $I_{max} = 50$. The percentage of valid results obtained for the second group of experiments was a bit higher than for the first one. We were expecting that the value for the second group of experiments could be lower because of the increased time to generate a valid solution.

Table 2. The results for four parameters

I_{max}	GW 2024					Random		
	m	w	I_b	I_a	% of winning solutions	I_b	I_a	% of winning solutions
5	20	4	**4**	**5**	**22**	–	–	0
10	20	4	**4**	**9**	**56**	–	–	0
20	20	4	**6**	**15**	**94**	18	19	10
50	20	4	**5**	**19**	**100**	18	27	98
5	25	5	**5**	**5**	**30**	–	–	0
10	25	5	**5**	**9**	**68**	–	–	0
20	25	5	**5**	**14**	**94**	20	20	4
50	25	5	**6**	**16**	**100**	19	25	100

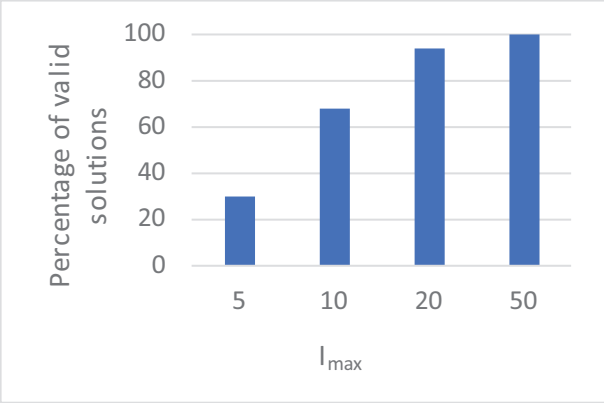

Figure 6. A percentage presentation of a number of valid results for the second set of parameters for the second group of experiments

However, it must be remembered that results obtained using the GWO algorithm are based on probability, and the number of valid solutions obtained for both groups of parameters is very similar. Only for $I_{max} = 5$, the difference of the value oscillates between 8–10%, but such a sharp constraint could not be a good reference point. What is worth underlining is that in most of the sets of the experiments, the winning combination of parameters was obtained in more than half of the runs. Therefore, even though one run of the algorithm did not produce a valid solution, there was a big chance that the second run would generate the winning combination.

5. Conclusion

In this paper, the grey-wolf optimization algorithm for the concurrent real-time optimization problem in game theory was presented. Due to our best knowledge, it is the first solution that deals with the problem of real-time optimization in game theory and the third area of usage of such optimization. The implemented algorithm is well known to be very fast because it narrows down the search space. It causes GWO also to be well known for stopping in local minima of optimizing parameters. However, the specification of the problem (more than one target vector of the parameters) decreases such a disadvantage.

Comparison of the results obtained by grey wolf algorithm with solutions obtained by random algorithm shows the effectiveness of chosen algorithm to investigated problem. Grey wolf algorithm was able not only to generate the results faster than random solutions but also gave valid results even when the second algorithm did not provide any valid result.

In the future, we will try to use some modifications of GWO and apply them to the problem discussed in this paper. It is also possible that applying another metaheuristic can give good results in solving the investigated problem. We will also check the efficiency of evolutionary computation in this problem in game theory. Concurrent real-time optimization is quite new problem in computer science. So far, it has been used to detect unexpected tasks in the IoT design process. Finding more areas of usage, this kind of optimization also seems to be a good direction of research.

AUTHORS
Adam M. Górski* – Institute of Applied Computer Science, Jagiellonian University, Poland, e-mail: a.gorski@uj.edu.pl.
Maciej Ogorzałek – Institute of Applied Computer Science, Jagiellonian University, Poland, e-mail: maciej.ogorzalek@uj.edu.pl.
*Corresponding author

References

[1] A. Górski and M. 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*, pp. 74–77, 2024. doi: 10.5281/zenodo.10046580

[2] A. Górski and M. Ogorzałek, "Concurrent Real-Time Optimization in Embedded System Design Process Using Genetic Algorithm", *5th Polish Conference on Artificial Intelligence*, Warsaw, Poland, Apr. 2024.

[3] X. Liang and Y. Xiao, "Game Theory for Network Security," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 472–486, 2013.

[4] D. Sanchez and H. Florez. "Improving Game Modeling for the Quoridor Game State Using Graph

Databases," in *Proceedings of the International Conference on Information Technology & Systems (ICITS 2018)*, Á. Rocha and T. Guarda, Eds., Advances in Intelligent Systems and Computing, vol 721. Cham: Springer, 2018. doi: 10.1007/978-3-319-73450-7_32.

[5] J. Konert, V. Wendel, S. Göbel, and R. Steinmetz, "Towards an Analysis of Cooperative Learning-Behaviour in Social Dilemma Games," in *Proceedings of the European Conference on Games-based Learning*, 2011, pp. 329–332.

[6] I. Grabska-Gradzińska, L. Nowak, and E. Grabska, "Towards an Analysis of Cooperative Learning-Behaviour in Social Dilemma Games," in *Artificial Intelligence and Soft Computing (ICAISC 2020)*, Lecture Notes in Computer Science, vol. 12415. Cham: Springer, 2020. doi: 10.1007/978-3-030-61401-0_37.

[7] F. Laamarti, M. Eid, and A. El Saddik, "An Overview of Serious Games," *International Journal of Computer Games Technology*, pp. 1–15, 2014. doi: 10.1155/2014/358152.

[8] A. Min, H. Min, and S. Kim, "Effectiveness of Serious Games in Nurse Education: A Systematic Review," *Nurse Education Today*, vol. 108, 105178, Elsevier, 2022.

[9] W. Johnson, "Serious Use of a Serious Game for Language Learning," *International Journal of Artificial Intelligence in Education*, vol. 20, 2010, pp. 175–195.

[10] M. Mortara, C. E. Catalano, F. Bellotti, G. Fiucci, M. Houry-Panchetti, and P. Petridis, "Learning Cultural Heritage by Serious Games," *Journal of Cultural Heritage*, 15(3), pp. 318–325, 2014.

[11] K. Madani, T. W. Pierce, and A. Mirchi, "Serious Games on Environmental Management," *Sustainable Cities and Society*, vol. 29, 2017, pp. 1–11.

[12] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," *Advances in Engineering Software*. vol. 69, pp. 46–61, 2014. doi: 10.1016/j.advengsoft.2013.12.007.

[13] D. Wang, D. Tan, and L. Liu, "Particle Swarm Optimization Algorithm: An Overview," *Soft Computing*, vol. 22, pp. 387–408, 2018. doi: 10.1007/s00500-016-2474-6.

[14] S. Saremi, S. Mirjalili, and A. Lewis, "Grasshopper Optimization Algorithm: Theory and Application," *Advances in Engineering Software*, vol. 105, pp. 30–47, 2017.

[15] S. Mirjalili and A. Lewis, "The Whale Optimization Algorithm," *Advances in Engineering Software*, vol. 95, pp. 51–67, May 2016.

[16] S. Mirjalili, "Dragonfly Algorithm: A New Meta-Heuristic Optimization Technique for Solving Single-Objective, Discrete, and Multi-Objective Problems," *Neural Computing and Applications*, vol. 27, no. 4, pp. 1053–1073, 2016.

[17] P. Hu, J. Pan, and S. Chu, "Improved Binary Grey Wolf Optimizer and Its Application for Feature Selection," *Knowledge-Based Systems*, Vol. 195, 105746, 2020, doi: 10.1016/j.knosys.2020.105746

[18] Y. Wang, T. Wang, S. Dong, and C. Yao, "Improved Binary Grey Wolf Optimizer and Its Application for Feature Selection," *Journal of Physics: Conference Series*, vol. 1682, Article 012020, 2020. doi: 10.1088/1742-6596/1682/1/012020

[19] A. Saxena, R. Kumar, and S. Das "β-Chaotic map enabled Grey Wolf Optimizer," *Applied Soft Computing*, vol. 75, Elsevier, pp. 84–105, February 2019.

[20] H. Yu, Y. Yu, Y. Liu, Y. Wang, and S. Gao, "Chaotic Grey Wolf Optimization," in *Proceedings of the 2016 International Conference on Progress in Informatics and Computing (PIC)*, Shanghai, China, 2016, pp. 103–113. doi: 10.1109/PIC.2016.7949476.

[21] K. Meidani, A. Hemmasian, S. Mirjalili, and A. B. Farimani, "Adaptive Grey Wolf Optimizer," *Neural Computing and Applications*, vol. 34, pp. 7711–7731, 2022. doi: 10.1007/s00521-021-06885-9

[22] W. Gu, "An Improved Multi-Objective Grey Wolf Optimization Algorithm with Dynamic Chaos Local Search Mechanism," in *Proceedings of the 2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, Chongqing, China, 2020, pp. 2020–2024. doi: 10.1109/ITAIC49862.2020.9338760

[23] S. Bahuguna and A. Pal, "β-Hill Climbing Grey Wolf Optimizer," in *Soft Computing for Problem Solving*, A. Tiwari, K. Ahuja, A. Yadav, J. C. Bansal, K. Deep, and A. K. Nagar, Eds., Advances in Intelligent Systems and Computing, vol. 1393. Singapore: Springer, 2021. doi: 10.1007/978-981-16-2712-5_26

[24] Q. Fan, H. Huang, Y. Li, Z. Han, Y. Hu, and D. Huang, "An Improved Multi-Objective Grey Wolf Optimization Algorithm with Dynamic Chaos Local Search Mechanism," *Expert Systems with Applications*, vol. 165, 2020.

[25] Y. Yang, B. Yang, S. Wang, W. Liu, and T. Jin, "An Improved Grey Wolf Optimizer Algorithm for Energy-Aware Service Composition in Cloud Manufacturing," *The International Journal of Advanced Manufacturing Technology*, vol. 105, pp. 3079–3091, 2019. doi: 10.1007/s00170-019-04449-9

[26] J. Jiang, Z. Zhao, Y. Liu, W. Li, and H. Wang, "DSGWO: An Improved Grey Wolf Optimizer with Diversity Enhanced Strategy Based on Group-Stage Competition and Balance Mechanisms," *Knowledge-Based Systems*, vol. 250, art. no. 109100, Aug. 2022.

[27] M. H. Nadimi-Shahraki, S. Taghian, and S. Mirjalili, "An Improved Grey Wolf Optimizer for Solving Engineering Problems," *Expert Systems with Applications*, vol. 166, 113917, 2021. doi: 10.1016/j.eswa.2020.113917

[28] J. Pan, Y. Gao, Q. Qian, Y. Feng, Y. Fu, M. Sun, and F. Sardari, "An Improved Grey Wolf Optimizer for Solving Engineering Problems," *Optik*, vol. 242, art. no. 167150, 2021, doi: 10.1016/j.ijleo.2021.167150

[29] F. A. Saif, R. Latip, Z. M. Hanapi, and K. Shafinah, "Multi-Objective Grey Wolf Optimizer Algorithm for Task Scheduling in Cloud-Fog Computing," *IEEE Access*, vol. 11, pp. 20635–20646, 2023, doi: 10.1109/ACCESS.2023.3241240

[30] S. Mangalampalli, G. R. Karri, and M. Kumar, "Multi-Objective Task Scheduling Algorithm in Cloud Computing Using Grey Wolf Optimization," *Cluster Comput.*, vol. 26, pp. 3803–3822, 2023. doi: 10.1007/s10586-022-03786-x

[31] C. Qu, W. Gai, M.Zhong, and J. Zhang, "A Novel Reinforcement Learning-Based Grey Wolf Optimizer Algorithm for Unmanned Aerial Vehicles (UAVs) Path Planning," *Applied Soft Computing*, vol. 89, 106099, 2020, doi: 10.1016/j.asoc.2020.106099

[32] J. Liu, X. Wei, and H. Huang, "An Improved Grey Wolf Optimization Algorithm and Its Application in Path Planning," *IEEE Access*, vol. 9, pp. 121944–121956, 2021, doi: 10.1109/access.2021.3108973

[33] A. M. Górski and M. Ogorzałek, "An Improved Grey Wolf Optimization Algorithm and Its Application in Path Planning," in *Proceedings of the 21st International SoC Design Conference (ISOCC)*, Sapporo, Japan, pp. 398–399, August 2024, doi: 10.1109/ISOCC62682.2024.10762129

[34] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: Task Graphs for Free," in *Proc. Workshop on Hardware/Software Codesign*, 1998, pp. 97–101.

[35] D. J. Abraham, K. Cechlárová, D. Manlove, and K. Mehlhorn, "Pareto Optimality in House Allocation Problems," in *Proc. 16th Int. Symp. Algorithms Comput. (ISAAC)*, Lecture Notes in Computer Science, vol. 3341, Springer, 2005, pp. 1163–1175.

[36] A. M. Górski, "Extended Task Graph for Real-Life Optimization Problems," in *Proceedings of the 7th International Conference on the Dynamics of Information Systems*, Lecture Notes in Computer Science, vol. 14661, Springer, 2025, pp. 74–86.