Autonomous Navigation Through the Maze Using Coevolution Strategy

Iaroslav Omelianenko[1], Anatoliy Doroshenko[1] and Yevheniy Rodin[1]

[1] *Institute of Software Systems of NAS of Ukraine, 40, Academician Glushkov avenue, Kyiv, 03187, Ukraine*

Abstract

This study explores the use of coevolutionary methods to address the challenge of navigating through complex maze environments with autonomous agents controlled by artificial neural networks. It underscores a critical impediment to algorithmic optimization: the close interdependence between the task's goal and the objective function used for optimal solution discovery. The task's goal is clear — identify the most efficient route through the maze. However, the objective function's formulation is more complex. In complex maze layouts, numerous deceptive areas may appear proximate to the exit but culminate in dead ends. Consequently, an elementary objective function that merely gauges the proximity to the exit can encounter numerous local optima within this deceptive search space, hindering the search for optimal solution. As maze complexity increases, such an objective function inevitably becomes ensnared in a local optimum, rendering the navigation issue unsolvable.

To counteract this, the study proposes a coevolution strategy involving a population of decision-making agents and a population of objective function candidates. This approach diverges from prior research by incorporating the NEAT algorithm to steer the coevolution of both populations. Additionally, the Novelty Search method was suggested to optimize the search within the potential solution space, favoring the most novel solutions.

The paper details the mathematical framework for crafting the objective function template, which integrates the novelty value of the discovered solution and its distance from the maze's exit. This framework serves as the foundation for defining the genomes of the organisms — candidates for the objective functions.

For comparison with preceding works, an experiment was conducted to evaluate the efficacy of the proposed coevolution method in resolving the problem of navigation within a complex maze environment.

Keywords

genetic algorithms, neuroevolution of augmenting topologies, autonomous maze navigation, NEAT, coevolution, CEUR-WS

1. Introduction

Application of any evolutionary algorithm (EA) to solve problem of navigation in a complex environment can be reduced to finding the optimal path allowing to reach the goal. This can be done by defining an objective function (fitness function) that we are going to minimize of maximize. However, in the previous work [1, 2] it was shown that there is a fundamental issue that arise in practice. While the objective of a task may be well defined and well known, the objective function may be deceptive.

We can easily illustrate the above statement on the example of building a model of an autonomous agent for solving the two-dimensional maze shown in Fig. 1. The task of the agent's control model is to steer the robot in such a way that it can go through the maze from the initial position to the exit in a specified number of steps. The control model determines the behavior of the

14th International Scientific and Practical Conference from Programming UkrPROGP'2024, May 14-15, 2024, Kyiv, Ukraine

yaric.mail@gmail.com (Iaroslav Omelianenko); doroshenkoanatoliy2@gmail.com (A. Doroshenko); yevheniy.s.rodin@gmail.com (Yevheniy Rodin)

0000-0002-2190-5664 (Iaroslav Omelianenko); 0000-0002-8435-1451 (Anatoliy Doroshenko; 0000-0003-2416-8572 (Yevheniy Rodin)

robot (direction of the movement) at each step depending on the current state of the environment: the position of the robot in the maze, the distance to the walls, and azimuth to the exit point.

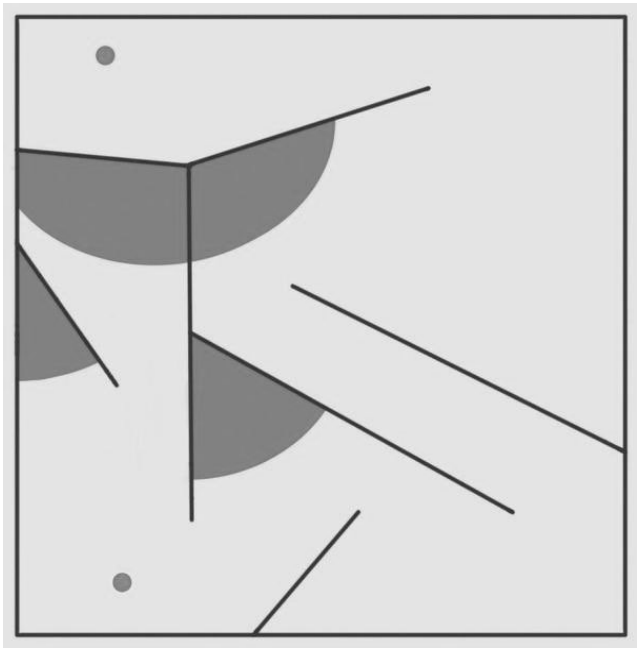

Figure 1: Two-dimensional maze with local optima cul-de-sacs (shadowed).

By intuition we can define the objective function as distance from the current position of the robot in the maze and maze exit point, as was done in [1]. However, in complex maze environment solver agent may face the increasing difficulty of choosing an adequate direction of movement based only on the distance to the exit.

As can be seen in Figure 1 even if the distance to the maze exit seems to be minimal, this doesn't mean that the path to the exit has been found. The objective function can have local optima of fitness scores in cul-de-sacs of the maze where steep gradients of fitness score values are registered [3, 4]. As a result, we have a deceptive landscape of fitness score values, that cannot be solved of a goal-oriented objective function.

Thus, we can see that even if optimal solution can be defined by control model, it cannot always be found during the evolutionary process using a simple objective function. Even more, our intuition about objective function seems flawed, as it equates the ultimate goal (maze exit) with the design of the objective function (distance to the maze exit).

As a solution for this problem, it was proposed in [5] to separate the optimization of the control model and the optimization of the objective function using the SAFE (Solution and Fitness Evolution) method. The crucial idea behind this method is following – even if the goal of the solver agent is to reach the maze exit point, this doesn't necessarily mean that the objective function is the distance to the exit. Thus, it was proposed to use the method of co-evolution of two populations of organisms – the population of control models and the population of candidates for objective functions.

The novelty of the approach considered in this paper is the use of the NEAT neuroevolution algorithm [2, 6, 7, 9] for the evolution of the population of controlling models.

The purpose of this work is the research and implementation of a new method of training of artificial neural networks (ANN) that can control the navigation of autonomous robotic systems in a complex environment.

2. Related works

Navigating optimally in complex environments is crucial for developing autonomous agents, and numerous scientific studies have tackled this challenge. In [1] it was proposed to use the NEAT algorithm with a simple objective function defined as the distance between the current position of

the agent and the final goal (exit from the maze). However, this approach is effective for simple maze layouts but encounters difficulties with more intricate maze configurations.

To address the challenge of developing optimal control models for complex maze configurations, [3, 4, 8] introduced an optimization method called Novelty Search (NS). This method rewards organisms that discover novel solutions within the space of previously found solutions, defining the novelty factor as objective function rather than the distance to the goal. The primary driver of NS is natural, unrestricted evolution, which often optimizes organisms' genomes by using new combinations to explore unknown territories and seek new behavioral patterns.

NS offers several advantages, including an unlimited search space and the potential to find non-trivial solutions. However, focusing solely on novelty can be problematic, as it doesn't incentivize a control model to stay on a goal-oriented trajectory and improve performance. Additionally, the time required to solve navigation problems using NS can be substantial, especially with complex maze configurations, making this method impractical in many cases.

Unlike the above-mentioned optimization methods, the approach of coevolving population of solver agent with population of candidates in the objective function, as proposed in this work, enables the discovery of optimal navigation solutions for complex maze configurations within a reasonable timeframe. Additionally, employing the NEAT algorithm to guide the evolution of solver agents results in optimal ANN topologies of control models. As a result, this enhances energy efficiency of produced control ANNs, making these models suitable for systems with limited computing and energy resources, such as industrial robots or autonomous drones.

The innovative aspect of the method discussed in this paper lies in the use of the NEAT algorithm for evolving the population of solver agents (control models) and the NEAT algorithm with NS-based search optimization for evolving the population of candidate objective functions. Furthermore, this work introduces a new modification of the NS method designed to limit the search space, thereby enhancing performance.

3. Background

The natural evolution of biological systems is intrinsically linked to the concept of coevolution. Coevolution serves as one of the primary driving forces in evolution. Its influence can be seen in the current state of the biosphere and the existing diversity of organisms.

Coevolution can be described as a mutually advantageous strategy where multiple genealogies of different organisms evolve simultaneously. The evolution of one species is inherently dependent on others. Throughout evolution, coevolving species interact, and these interspecies relationships shape their evolutionary strategies. We can define three main types of the coevolution:

- mutualism, when two of more species peacefully coexist and receive mutual benefit from each other,
- concurrent coevolution:
  a) predation, when one organism kills another and consumes its resources,
  b) parasitism, when one organism uses the resources of another, but does not kill it,
  - predation, when one organism kills another and consumes its resources,
  - parasitism, when one organism uses the resources of another, but does not kill it,
  - commensalism, it occurs when members of one species gain benefits without affecting the other species, either positively or negatively.

The last type of coevolutionary strategy has attracted the attention of researchers in [5] due to its potential for developing an effective method for training autonomous agents. As a result, the implementation of the SAFE algorithm was proposed, which we will consider further.

4. SAFE algorithm features

As the name suggests, the SAFE coevolution method of the solver and fitness function relies on their simultaneous evolution, guiding the optimization of the solution search. The SAFE method is centered on the strategy of commensalistic coevolution between two populations:

a population of potential solutions that evolve to solve the task,

a population of candidate objective functions that evolve to guide the evolution of the population of solutions.

The core concept of the SAFE algorithm is to leverage distinct search optimization methods for each of the two populations independently. Objective-based optimization, which measures the distance to the maze exit, can be used to estimate the fitness score of each solution in the population of potential solutions. NS-based optimization is suitable for guiding the evolution of population of candidates for objective functions. With NS optimization, we are not focused on a specific metric but rather on exploring various paths in the solution search space.

This work suggests using the NEAT algorithm to manage the evolution of a population of potential solutions, combined with NS (novelty search) to enhance the evolutionary process in the population of candidates for objective functions.

Next, we consider a modified maze experiment that will be used to assess the effectiveness of the proposed solution.

5. Maze experiment

In [1], was presented how to use the NEAT algorithm to address the classic problem of navigating through a maze. In this experiment, a straightforward objective function, based on the distance from the agent's current position to the maze exit, was used to optimize the search.

The experiment is built around the robot which navigates the maze using information about environment acquired from its sensors, see Figure 2.

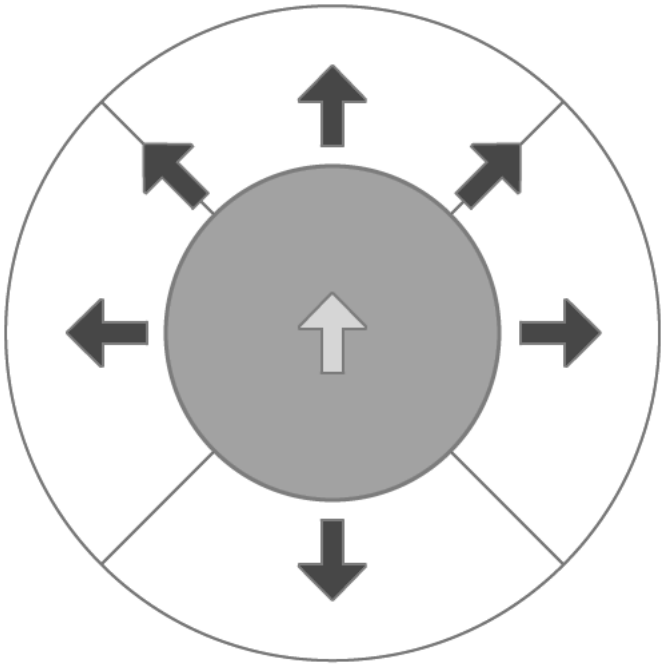

Figure 2: The configuration of sensors of the robot.

In Figure 2, the robot's rigid body is depicted as a filled circle, with an arrow inside indicating its heading. The six surrounding arrows represent rangefinder sensors that measure the distance to the

nearest obstacle in their respective directions. Additionally, the four outer circle segments illustrate the four pie-slice radar sensors, which function as a compass guiding the robot to the maze exit.

Each radar sensor activates when the line from the maze exit to the robot's center falls within its field of view (FOV). The detection range of the radar sensor is confined to the maze area within its FOV. Consequently, at any given moment, one of the four radar sensors is active, indicating the direction of the maze exit.

The FOV of each radar sensor presented in Table 1.

Table 1
Radar sensors FOV

Sensor	FOV, degrees
Front	315.0 ~ 405.0
Left	45.0 ~ 135.0
Back	135.0 ~ 225.0
Right	225.0 ~ 315.0

The rangefinder sensor is a ray projected from the robot's center in a specific direction. It activates upon intersecting with an obstacle and measures the distance to it. The detection range of this sensor is determined by a specific configuration parameter of the robot-navigator. Table 2 presents the directions that the rangefinder sensors monitor relative to the robot's heading.

Table 2
Rangefinder sensors monitor directions

Sensor	Direction, degrees
Front	0.0
Front-Right	-45.0
Right	-90.0
Front-Left	45.0
Left	90.0
Back	-180.0

The robot's movement is controlled by two actuators that apply forces to turn and/or propel the agent's frame, altering its linear and/or angular velocity.

In Figure 1 depicted the maze configuration that is used in the experiment. The maze is an area enclosed by outer walls. The entry point of the maze at the bottom-left and exit point at the top-left. Inside maze area, various internal walls create multiple dead ends with local fitness optima (shadowed), making objective-oriented optimization less effective. Additionally, due to these local fitness optima, objective-based search agents can become trapped in a specific dead end, completely halting the evolution process.

As previously mentioned, the SAFE method involves two distinct evolutionary processes: one for the population of potential solutions and another for the population of candidates for the objective function. Therefore, we need to define the fitness functions for each evolutionary process.

6. Maze solver fitness function

In each generation of the evolution, every solution individual (maze solver) is evaluated against all objective function candidates. The fitness score of the solution is determined by the highest

fitness score obtained during these evaluations. The fitness function of the maze solver combines two metrics: the distance from the maze exit (objective-based score) and the novelty of the solver's final position (novelty score). These scores are arithmetically combined using a pair of coefficients derived from the specific individual in the population of the objective function candidates as can be seen from the following formula:

$$O_i(S_i) = a \times \frac{1}{D_i} + b \times NS_i, \quad (1)$$

where $O_i(S_i)$ – is the fitness score values from obtained by evaluating the solution candidate S_i against the objective function O_i. Here, the pair of coefficients, [a, b], is the output of the specific objective function candidate. This pair determines how the distance to the maze exit (D_i) and the behavioral novelty (NS_i) of the solution influence the final fitness score of the maze solver at the end of its trajectory.

The distance to the maze exit (D_i) is calculated as the Euclidean distance between the maze solver's final coordinates and the maze exit coordinates. This is represented by the following formula:

$$D_i = \sqrt{\sum_{i=1}^{2}(a_i - b_i)^2}, \quad (2)$$

where a is the coordinates of the maze solver at the end of its trajectory and b is the maze exit coordinates.

The novelty score NS_i of each maze solver is based on its final position in the maze (point x). It is calculated as the average distance from this point to the k -nearest neighbor points, which are the final positions of the other maze solvers. The novelty score value at point x of the behavioral space can be calculated with following formula:

$$NS_i = \frac{1}{k}\sum_{i=0}^{k} dist(x, \square_i), \quad (3)$$

where $\square_i$ is the i-th nearest neighbor of x and $dist(x, \square_i)$ is the distance between $\square_i$ and x.

The novelty metric, which measures how different the current solution (x) is from another ($\square_i$) produced by different maze solvers, is calculated as the Euclidean distance between the two points by formula:

$$dist(x, \square) = \sqrt{\sum_{j=1}^{2}\left(x_j - \square_j\right)^2}, \quad (4)$$

where $\square_j$ and x_j are the values at position j of vectors holding the coordinates of points $\square$ and x.

7. Objective function candidates' fitness function

The SAFE method employs a commensalistic coevolutionary approach, where one of the co-evolving populations is neither benefited nor harmed during evolution. In an experiment, this commensalistic population consists of candidates for target functions. For these candidates, we need to define a fitness function that is independent of the quality of the maze-solving agents (controlling models).

A suitable option here is a fitness objective function that uses a novelty metric to estimate fitness. The formula for calculating the novelty score of each objective function candidate is the same as for the maze-solving agents (3). The only difference is that, for objective function candidates, we calculate the novelty score using vectors with the output values of each organism in the population [a, b] (see equation 1). We then use the novelty score value as a measure of the organism's fitness.

In [3, 4, 8], an implementation of the NS optimization method is presented, where modifications were forcibly made to limit the search space. This was done to enhance the algorithm's performance and ensure its execution within a specified time frame.

8. Novelty Search modifications

The modifications to the NS method presented in this experiment involve a new approach to maintaining the archive of novelty points. An individual novelty point records the maze solver's location at the end of its trajectory, combined with its novelty score. In the traditional NS method, the size of the novelty archive is dynamic, allowing the addition of a novel point if its novelty score exceeds a certain threshold (the novelty threshold). This threshold can be adjusted during runtime based on the rate at which new novelty points are discovered, helping control the archive's maximum size. However, selecting an initial novelty threshold value is not straightforward.

The modified NS method introduces a fixed-size novelty archive to address the challenge of choosing the correct novelty threshold value. New novelty points are added to the archive until it is full. After that, a novelty point is only added if its novelty score exceeds the current minimum score in the archive, replacing the point with the lowest score. This approach maintains a fixed archive size, storing only the most valuable novelty points discovered during the evolution.

9. Maze experiment

In this experiment, we need to establish two co-evolving populations with different initial genotype configurations to meet the phenotypic requirements of the resulting species. The maze solver's phenotype includes 11 input nodes for sensor signals and two output nodes for control signals. Meanwhile, the objective function candidate's phenotype has one input node receiving a fixed value (0.5), which is converted into two output values used as the fitness function coefficients for the maze solver.

Additionally, we need to develop a maze-solving simulation environment to evaluate the solver agents at each epoch of the evolution.

10.Maze-solving simulation environment

In this work, software was developed to model the maze-solving problem, like the one described in [1], but using the GO programming language with modifications to facilitate the coevolution of two populations. The software comprises the following main components, each implemented as a separate class:

1. *Agent* – a class that stores information related to the maze navigator agent involved in the simulation.
2. *RecordStore* – a class that manages the storage of records related to the evaluations of all decision agents during the evolutionary process. The collected records can be used to analyze the evolutionary process after its completion.
3. *Environment* – a class that contains information about the maze simulation environment. This class also includes methods to control the maze simulation environment and the position of the solving agent, detect collisions with maze walls, and generate input data for the agent's sensors.
4. *NoveltyItem* – a class designed to encapsulate information about specific data item, that contains information about the novelty score associated with a particular genome, enhanced by supporting information. It is used in conjunction with *NoveltyArchive*.
5. *NoveltyArchive* – a class that manages the storage of all found *NoveltyItems* and provides methods to evaluate their novelty as new items are found.

Next, we are going to describe the experiment details.

11.Maze experiment details

This research aimed to develop a control ANN model capable of navigating a complex maze with configuration as illustrated in Figure 1. This allows for a comparison between the coevolution method and the simple evolution method described in previous work [1]. It has been demonstrated that while a simple evolutionary process controlled by the NEAT algorithm can solve basic maze navigation problems, it struggles with more complex maze configurations. Therefore, this experiment is crucial for demonstrating the effectiveness of the coevolution method.

The maze features two fixed positions marked by filled circles. The lower left circle indicates the starting position of the maze-solving agent, while the upper left circle marks the exact location of the maze exit. To complete the task, the robot must reach a point near the maze exit for specific number of time steps.

12.Maze experiment results

The experiment was conducted using GO programming language version 1.24.1, with the goNEAT version 4.0.2 [10] and goNEAT_NS version 4.0.2 [11] libraries used to develop the simulation and evaluation source code. The workstation used had the following specifications: CPU 2.3 GHz 8-Core Intel Core i9, 16 GB 2667 MHz DDR4, running macOS 14.4.1.

The successful evolutionary process runner has the NEAT hyper-parameters as specified in Table 3.

Table 3
NEAT hyper-parameters of successful evolutionary process

Parameter	Value
trait_param_mut_prob	0.5
trait_mutation_power	1.0
weight_mut_power	0.8
disjoint_coeff	1.0
excess_coeff	1.0
mutdiff_coeff	3.0
compat_threshold	6.0
age_significance	1.0
survival_thresh	0.2
mutate_only_prob	0.3
mutate_random_trait_prob	0.1
mutate_link_trait_prob	0.1
mutate_node_trait_prob	0.1
mutate_link_weights_prob	0.9
mutate_toggle_enable_prob	0.1
mutate_gene_reenable_prob	0.05
mutate_add_node_prob	0.03
mutate_add_link_prob	0.1
mutate_connect_sensors	0.5
interspecies_mate_rate	0.001
mate_multipoint_prob	0.6
mate_multipoint_avg_prob	0.4
mate_singlepoint_prob	0.0
mate_only_prob	0.2
recur_only_prob	0.2

pop_size	300
dropoff_age	200
newlink_tries	40
print_every	100
babies_stolen	0
num_runs	1
num_generations	2000
log_level	Info
epoch_executor	sequential
genome_compat_method	linear

After 211 generations of evolution, a successful solver agent was discovered, featuring a configuration of 22 nodes interconnected by 47 links.

During the coevolution process, the optimal coefficients [a, b] for the objective function were found and used to train a successful agent-solver: [-0.53283, 0.95889]. Thus, formula (1) can be rewritten by substituting these coefficients as follows:

$$O_i(S_i) = -0.53 \times \frac{1}{D_i} + 0.96 \times NS_i, \quad (5)$$

Based on formula (5), it can be concluded that the identified objective function prioritizes training to find the most innovative solutions (NS_i), while placing less emphasis on the distance to the maze exit (D_i). This supports the initial thesis of the paper, which states that in a complex environment, a successful objective function differs from merely minimizing the distance to the goal (exit from the maze).

It is worthwhile to examine the visualization of the maze traversal by the successful solver agent (Figure 3), noting the smoothness and closeness to the optimal of the route discovered. The path includes 400 recorded points marking the agent's location throughout the simulation.

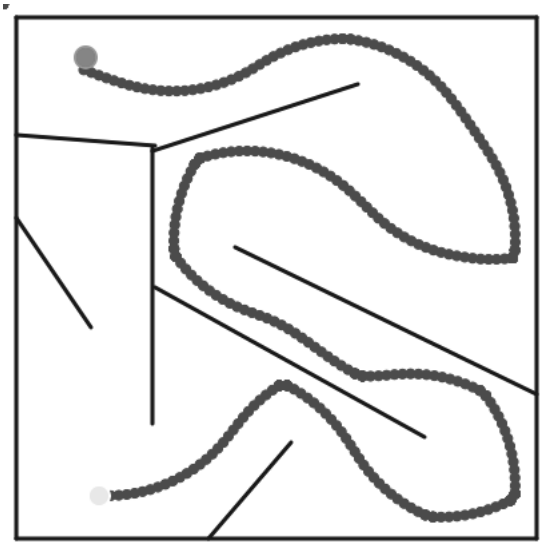

Figure 3: The path of the successful solver agent through the maze.

Additionally, as shown in Figure 4, throughout all epochs of evolution, only one species out of the 46 available generated the configuration of the controlling ANN for the successful agent-solver. This figure also illustrates that the most of evolutionary losers were trapped in the local optima. Meanwhile, the successful agent demonstrated the highest level of innovation, favoring the exploration of new areas over the exploitation of known ones (exploration vs. exploitation).

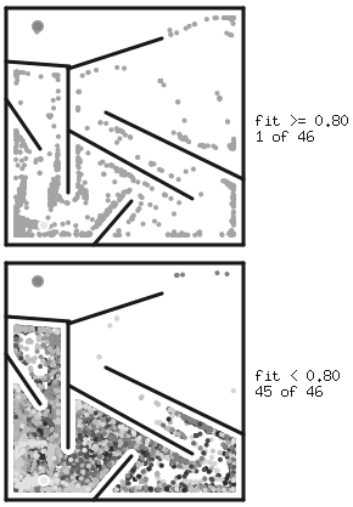

Figure 4: Visualization of evaluation of population of solver agents by species.

In Figure 4, to enhance the informativeness of the visualization, we introduced a fitness threshold to filter out the top-performing species. The top subplot displays the final positions of the solver agents from the champion species, with fitness scores exceeding 0.8. While bottom one depict the final positions of solver agents from species – evolutionary losers.

Conclusions

This paper demonstrates the application of coevolving two populations – a population of decision-making agents and a population of objective function candidates – to address the challenge of navigating a complex maze environmant. Experimental results obtained have demostrated that this method is more effective than the one discussed in [1].

Building on previous work [5], a novel approach to implementing the SAFE (Solution and Fitness Coevolution) algorithm was proposed. This approach involves using the NEAT algorithm to manage the evolutionary process of both populations linked by commensalistic coevolution. Additionally, the Novelty Search method was employed to optimize the search for solutions within a deceptive environment of potential solutions.

As result of this research, the software was developed using the GO programming language [12] to conduct the coevolution experiment. Additionally, advanced visualization tools were created to visually assess the results of coevolution as input parameters change.

This method of training controlling ANNs can be applied to create autonomous robotic systems (drones) capable of navigating complex environments, making it relevant for both humanitarian demining and other specialized tasks.

References

[1] Omelianenko, Iaroslav. Simulation of the autonomous maze navigation using the NEAT algorithm. Problems in programming 4, (2023), 76-89. doi:10.15407/pp2023.04.076

[2] Omelianenko, Iaroslav. Hands-On Neuroevolution with Python: Build high-performing artificial neural network architectures using neuroevolution-based algorithms. Birmingham, UK: Packt Publishing Ltd, 2019. ISBN: 9781838824914, 368 p.

[3] J. Lehman, K.O. Stanley. Revising the evolutionary computation abstraction: minimal criteria novelty search. In Proceedings of the 12th annual Genetic and Evolutionary Computation Conference (GECCO-2010). ACM, 103–110. doi:10.1145/1830483.1830503

[4] J. Lehman, K.O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. Evolutionary computation 19, 2 (2011), 189–223. doi:10.1162/EVCO_a_00025

[5] M. Sipper, J. H. Moore and R. J. Urbanowicz. Solution and Fitness Evolution (SAFE): A Study of Multiobjective Problems. In proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC), Wellington, New Zealand, 2019, pp. 1868-1874, doi:10.1109/CEC.2019.8790274.

[6] K. O. Stanley, R. Miikkulainen. Evolving neural networks through augmenting topologies. Evolutionary computation 10, 2 (2002), 99–127. doi:10.1162/106365602320169811

[7] I. Sinitsyn, A. Doroshenko, T. Mamedov, O. Yatsenko. A method of automated design of neuroevolution algorithms based on Glushkov algebra of algorithms. International Scientific Technical Journal" Problems of Control and Informatics", 68, 3 (2023), pp. 74–85. doi:10.34229/1028-0979-2023-3-8

[8] Omelianenko, Iaroslav. Creation of Autonomous Artificial Intelligent Agents Using Novelty Search Method of Fitness Function Optimization. hal.science, 2018, url: https://hal.science/hal-01868756.

[9] Omelianenko, Iaroslav. Autonomous Artificial Intelligent Agents. In: Machine Learning and the City. John Wiley & Sons, 2022. Chap. 12, pp. 263–285. ISBN: 9781119815075, doi:10.1002/9781119815075.ch21

[10] Omelianenko, Iaroslav (2023). The GOLang implementation of NeuroEvolution of Augmenting Topologies (NEAT) algorithm (v4.0.2). [Computer software]. Zenodo. doi:10.5281/zenodo.10119451

[11] Omelianenko, Iaroslav (2024). The GOLang implementation of NeuroEvolution of Augmenting Topologies (NEAT) with Novelty Search optimization (v4.0.2). [Computer software]. Zenodo. doi:10.5281/zenodo.10951521

[12] Cox, R., Griesemer, R., Pike, R., Taylor, I. L., & Thompson, K. The Go programming language and environment. Communications of the ACM, 65, 5 (2022), pp. 70–78. doi:10.1145/3488716