The System for Finding the Least Resource-Intensive Path in Two- or Three-Dimensional Space Using Machine Learning

Vadim Kholoshnia[0000−0001−6485−6340] and Elena Boldyreva[0000−0001−6357−4448]

ITMO University. 197101, St.Peterburg, Russia
vdkholoshnia@gmail.com
eaboldyreva@itmo.ru

Abstract. In the course of the research work, the system for finding the least resource-intensive path in two- or three-dimensional space, based on machine learning with visualization of the algorithms execution process was developed. The universality of the system lies in the fact that the user can determine the script for its execution - choose the optimal learning algorithm, load or create the desired map of the area and set the logic of the object movement. To demonstrate these features of the system software implementation, a user interface was developed. In order to formulate user recommendations and make it easier for the user to choose a machine learning algorithm, it was decided to develop a software implementation for visualizing the learning process of the genetic algorithm and reinforcement learning. In relation to the task of finding the least resource-intensive path in two- or three-dimensional space, the genetic algorithm and reinforcement learning were tested and compared. Based on a comparative analysis, recommendations on the choice of a learning algorithm are formulated.

Keywords: least resource-intensive path finding · shortest path finding · machine learning · genetic algorithm · reinforcement learning · visualization of learning

1 Introduction

Currently, there is an increasing need to create systems for automatically finding the least resource-intensive path in two or three dimensional spaces for objects without human intervention. One way to achieve this goal is to use machine learning methods such as genetic algorithm and reinforcement learning. Reinforcement learning is one of the machine learning methods, during which the test system (agent) learns by interacting with a certain environment. The genetic algorithm is a heuristic search algorithm used to solve optimization and modeling problems by randomly selecting, combining, and varying the desired parameters using mechanisms similar to natural selection in nature. These algorithms allow building paths in space, having information only about the environment. Also, such algorithms are universal, since the ability to configure allows using them almost on any map and in any conditions.

The purpose of this study is to reduce the cost of resources for finding and traversing a path for an object with arbitrary logic of movement on any map. To achieve this goal, it was decided to develop the least resource-intensive path finding algorithm based on the NeuroEvolution of Augmenting Topologies (NEAT) [1] genetic algorithm and its software implementation using multi-threaded programming technology. For the case when the user needs to get information for all the least resource-intensive paths in space for any possible starting point it was decided to use Q-Learning [2] reinforcement learning algorithm. The least resource-intensive path means not only the shortest path from the starting point to the goal, but also the path during which the object uses the least amount of resources (e.g. the time required to complete the path).

The novelty of the results is that, in contrast to existing systems, the developed least resource-intensive path finding system is adaptable. I.e the script for the execution of the algorithms can be adjusted by the user by choosing the optimal algorithm for training the neural network model, selecting criteria for selecting the optimal path, creating the necessary map and determining the logic of the object movement. The least resource-intensive path finding algorithm based on the reinforcement learning (Q-Learning), unlike the existing ones, is implemented for training on a map of the area, presented in the form of a spatial grid of possible states or obstacles, in the form of a map that the user can specify. The visualization of the machine learning processes can not only aid the process of formulating user recommendations but also it allows using presented system for educational purposes, such as creating virtual data science laboratory. In addition, the software implementation of the developed system allows a comparative analysis of used machine learning algorithms, which, of course, is of scientific value.

2 Literature review

In the context of this study, it was decided to consider the NEAT genetic algorithm and Q-Learning reinforcement learning as the main machine learning algorithms.

The NeuroEvolution of Augmenting Topologies learning method (trial and error) is less resource-intensive (less computational power usage), but no less accurate when constructing a path. Also, this algorithm often reaches effective networks faster than other modern neuroevolutionary methods and reinforcement learning methods [3, 4].

The Q-Learning learning method (reinforcement learning algorithm) is more resource-intensive when learning (more computational power usage), but the result of this algorithm is an array of possible states and actions that allows choosing the initial position of the object after training.

In addition, the analysis of existing algorithms and implementations of the least resource-intensive path finding has been carried out. Due to the fact that there are many different formulations of this problem, there are a lot of algorithms for solving the shortest path finding problem, such as the Dijkstra algorithm [5], the Bellman-Ford algorithm [6], the A* algorithm [7], the Floyd-Warshell algorithm [8], the Johnson algorithm [9], the Lee algorithm [10] and others. Unlike the presented shortest path finding methods, these algorithms assume the presence of an already generated weighted graph or they build a path to the goal for each initial state. Also, these algorithms do not take into account the features of the logic of the object movement.

3 Content of research

To develop and demonstrate the effectiveness and value of the least resource-intensive path finding system, the following steps must be completed:

1. To develop a functional structure (implementation) of the least resource-intensive path finding system;
2. To adapt and to refine the presented system depending on the chosen learning algorithms - NEAT and Q-learning;
3. To develop a software implementation of the shortest path finding algorithm;
4. To develop a user interface for software implementation;
5. To analyze the results of the software implementation of the least resource-intensive path finding system and used machine learning algorithms.

3.1 Development of the functional structure of the algorithm

At this stage, the least resource-intensive path finding system was formed based on the above machine learning methods. The algorithm diagram is shown in Figure 1.

After launching the presented system, it becomes necessary to choose one of the learning algorithms. In the context of this study, it was decided to consider the NEAT genetic algorithm and Q-learning reinforcement learning as the main machine learning algorithms, since each of these algorithms optimally functions only under certain conditions, which together cover the maximum number of possible conditions of user needs.

3.2 Adaptation of the shortest path finding algorithm depending on the chosen learning algorithm

Implementation of the least resource-intensive path finding algorithm using the genetic algorithm (NEAT)

The least resource-intensive path finding algorithm using the genetic algorithm (NEAT) works according to the following principle: when the program launches,

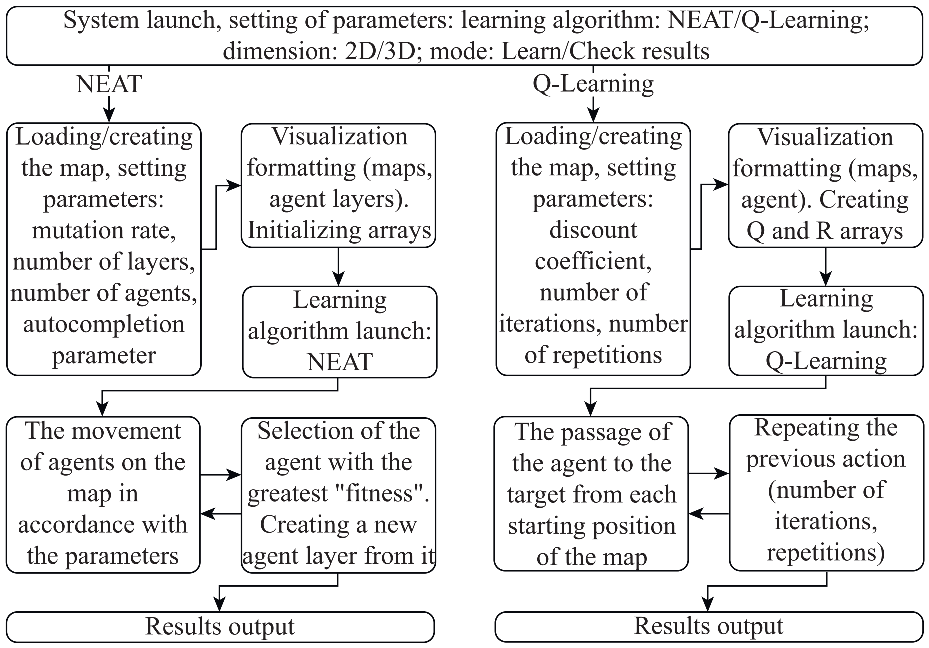
System launch, setting of parameters: learning algorithm: NEAT/Q-Learning; dimension: 2D/3D; mode: Learn/Check results
NEAT
Q-Learning
Loading/creating the map, setting parameters: mutation rate, number of layers, number of agents, autocompletion parameter
Visualization formatting (maps, agent layers). Initializing arrays
Learning algorithm launch: NEAT
The movement of agents on the map in accordance with the parameters
Selection of the agent with the greatest "fitness". Creating a new agent layer from it
Results output
Loading/creating the map, setting parameters: discount coefficient, number of iterations, number of repetitions
Visualization formatting (maps, agent). Creating Q and R arrays
Learning algorithm launch: Q-Learning
The passage of the agent to the target from each starting position of the map
Repeating the previous action (number of iterations, repetitions)
Results output

Fig. 1. Software implementation scheme

each agent randomly generates an array of directions, presented in the form of positions for movements formed from angles (the size of this array can be adjusted for example to limit number of possible movements of the agents). After that, each agent starts moving in accordance with the directions array elements and stops moving as soon as it touches a user-defined area (e.g. an obstacle on the map), reaches the goal, or when the elements in the directions array end (the number of moves ends). Then, the selection (natural selection principle) of the best agents for further training occurs: the formation of the next generation based on data of the previous generation best agent and the mutation rate parameter happens. The larger this parameter is, the larger the array of directions of the next generation agents differs from the previous one. The "fitness" of an agent is set according to a certain formula, for example, for the presented algorithm, the "fitness" of the agent that is closest to the target as compared to the others is greater (1).

$$F = 1/dist(g, a)^2 \tag{1}$$

where F - the "fitness", g - the goal position, a - the agent position, dist(g, a) - the distance between goal and agent. If one of the agents has reached the goal, then for the agents of the next generation, the "fitness" of the one that has done the least number of moves is greater (2).

$$F = d + 1/s^2 \tag{2}$$

where F - the ”fitness”, d - the directions array size, s - the number of steps done before reaching the goal. The agents with the highest ”fitness” are remembered as the best.

The process of executing the algorithm can be divided into peculiar stages - generations. The more generations the algorithm goes through, the more accurate result (less resource-intensive path) will be. After determining the ”fitness”, the agents of the next generation receive the properties of the best agent from the previous one, but with changes obtained during selection depending on mutation rate parameter. The first stage (generation) ends here. This process is repeated for future generations, until the goal is achieved. After reaching the goal, the “fitness” formula of the agent changes, now the agent is considered the best, which has done the least number of moves (moves) before reaching the agent (after reaching the goal with at least one agent, the transition to the next generation takes place).

The presented algorithm also offers the possibility of using several layers of agents. Each level is independent of the others, which allows us to consider more possible ways. The layer with the greatest “fitness” of the best agent is considered the best. It is proposed to use this feature when there are several possible ways to build a path from the starting point to the goal on the map.

The execution of an algorithm ends automatically according to the autocompletion parameter. This parameter shows how many generations agents must go through after reaching the goal. Also, the user can interrupt the execution and save the results at any time, if necessary.

Implementing an algorithm for finding the least resource-intensive path using reinforcement learning (Q-Learning)

When the reinforcement learning algorithm is launched, an array of possible states and actions R is created, which shows where the agent can go and where not, as well as the location of the target on the map (numbering starts at zero and runs horizontally and top down in ascending order). After that, the second array Q is created and filled with zeros.

Then training takes place: in order for an agent to learn how to find the least resource-intensive path from any position on the map, it needs to check each initial state (after creating the first array R, an array of possible initial states is also created). Then the agent leaves each state and calculates his next move, comparing the rewards for all possible subsequent actions. After that, the second array with award weights is filled in accordance with formula (3) (Markov decision process [11]):

$$Q(s,a) = r(s,a) + \gamma max Q(s', a') \tag{3}$$

where s - the current state, a - the next action, γ - the discount coefficient. It is used to balance the immediate and future rewards. Typically, this value ranges from 0.8 to 0.99. The discount coefficient controls the speed of learning, the more this parameter is, the faster the neural network learns, but the more errors it can make. s' is the next state after the next action. a' is the next action after the next state. The maxQ(s', a') function returns the maximum reward for an available action from state s'. After the training is completed, the second array Q stores the rewards of the possible actions for each state.

The user can enter any initial state and get the least resource-intensive path. Such an algorithm can be used for devices that operate on static map, for example, a robot vacuum cleaner. At the first scan of the apartment, the algorithm will compose the R array and will be trained, and on subsequent trips, the vacuum cleaner will already have a ready array of Q rewards in it that will allow it to instantly get the least resource-intensive path to the goal (dock-station) from any starting location, which will also improve with every departure.

Based on the results of this stage, it was decided, when solving the problem of finding the least resource-intensive path, to use both described learning algorithms, since in the aggregate they cover almost all possible path finding needs and environmental conditions.

3.3 Software implementation of the algorithm

For visualization of the menu it was decided to use C++ Windows Forms API.

The proposed the least resource-intensive path finding algorithm on a two-dimensional map is described in the C++ programming language. To visualize the operation of the algorithm, the SFML graphical interface library was used (Fig. 2).

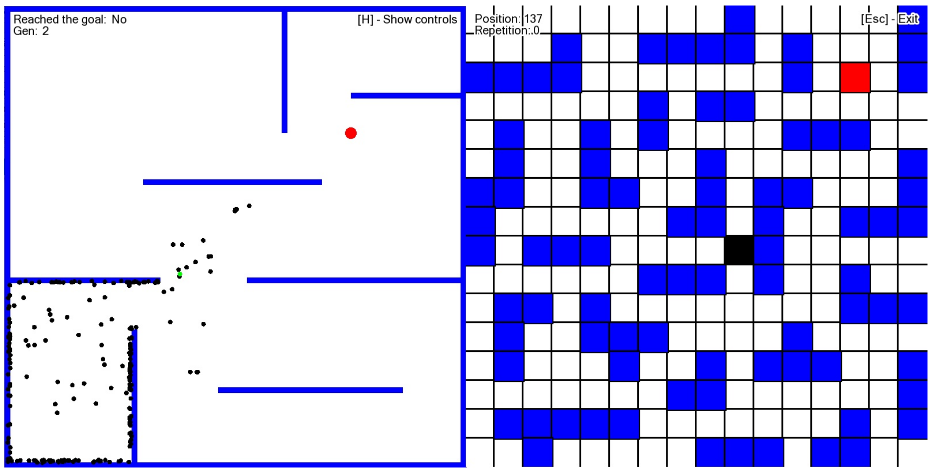

Fig. 2. Visualization of the learning algorithm on a two-dimensional map

To visualize a three-dimensional map, the Unity engine was used. The algorithm is described in the C# programming language (Fig. 3).

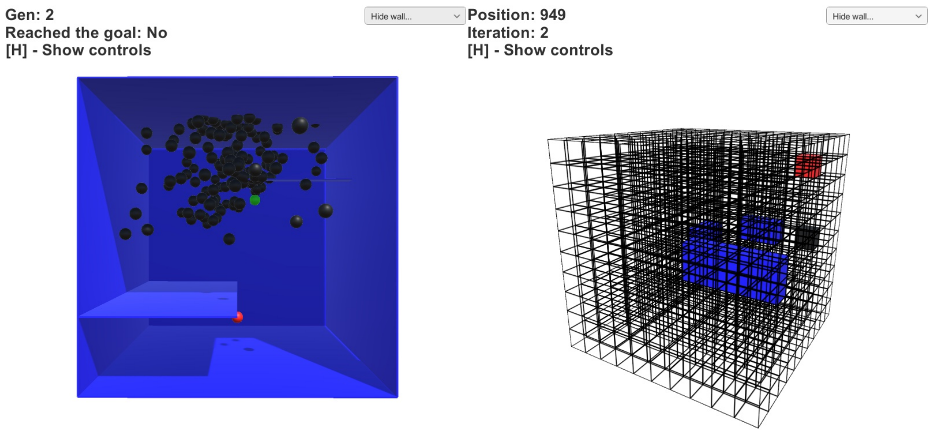

Fig. 3. Visualization of the learning algorithm on a three-dimensional map

Also, an algorithm for creating a two-dimensional and three-dimensional map is applied to the software implementation, which allows creating the necessary environmental conditions, load a map from a graphic format file (supported formats: BMP, PNG, TGA, JPG, GIF, PSD, HDR, PIC) and load 3D objects to use them when training (supported formats: FBX, DAE, 3DS, DXF, OBJ, SKP). The CSV extension data files, if necessary, simplifies editing with table editors.

To represent dynamic arrays, pointers and multithreading, the standard libraries were used, the use of which avoids possible errors and memory leaks.

Unity.Jobs was used to distribute processor power, as this library not only monitors all the little things and errors when working with threads, but also provides maximum performance when using multi-threaded programming technology in the Unity engine.

Implementation in C-like languages, as well as splitting the software implementation of the algorithm into separate files, which allows selecting parts with a description of only the desired algorithm, simplifies the process of embedding it into other systems.

For the genetic algorithm, the user can choose the speed of training and, accordingly, obtain the desired result. The speed and accuracy of training can be regulated by changing the number of agents, the more agents, the higher the learning speed, as long as the number of agents does not require more computing power. A used multithreading programming technology allows separate processor power to different streams and use each of them for each layer so as long as number of layers is less than supported streams by processor there won't be any performance issues.

For reinforcement learning, the user can select the learning speed by adjusting the parameter γ. The larger the parameter, the higher the learning speed, but also the probability of missing a useful learning outcome. To implement the ability to use the spatial grid as a map, an algorithm for representing the map in the form of a two-dimensional array of possible states and actions was added.

3.4 Analysis of the results

Based on the results of the software implementation, a comparative analysis of two training algorithms was carried out. The analysis data are presented in Table 1.

Table 1. Comparative analysis of the presented machine learning algorithms

	Q-learning (map resolution: 25x25)	NEAT (map resolution: 80x80)
Ability to choose an arbitrary starting point after training	+	-
Adjustable parameters	Obstacles on the map, map resolution, discount coefficient, number of iterations, number of repetitions	Obstacles on the map, map resolution, mutation rate, number of layers, number of agents, size of directions array, autocompletion parameter

The complexity of the map affects the runtime of the reinforcement learning algorithm: if number of the obstacles on the map increasing, the execution time decreases, since it decreases as the number of possible initial positions of the agent. Due to the structure of the genetic algorithm, the complexity of the card increases the learning time, but with the correct settings, the increase in the running time of the algorithm can be almost completely avoided.

Thus, the decision on the appropriateness of sharing various training algorithms within the same the least resource-intensive path finding system can be considered justified, since the user can choose a training algorithm for certain environmental parameters (map). Based on the results of the comparative analysis, the following recommendations were formed:

1. If it is necessary to obtain an exact path for a specific environment, it is proposed to choose the genetic algorithm NEAT, the result of which will be an array of object movements for a specific starting point. But when using the NEAT training method with the presented “fitness” formula, it is necessary to take into account that the initial position should not be beyond the obstacle, for which the agent would need to go a greater distance than the possible deviation, regulated by the mutation rate parameter. To solve this problem, it is necessary to increase the mutation rate parameter, which can

lead to a loss of accuracy for the same period of training time or, for example, use a system of additional rewards placed on the map and indicating to the agent how to get around the barrier. Also using this logic, the user can adjust the trajectory, if it is necessary.

2. When using training with Q-Learning reinforcement learning as a result, the user receives a two-dimensional array of possible states and a certain reward for actions that allows selecting any available starting point on the map after training. However, to obtain a more accurate result, more resources will be required (it is necessary to increase the number of map fields, which entails an increase in the amount of memory used).

4 Conclusion

In the course of research work:

- The system for finding the least resource-intensive path in two or three dimensional space using machine learning is developed. The universality of the algorithm lies in the fact that the user can choose the optimal learning algorithm, create the necessary map of the area and set the parameters.
- An effective software implementation has been developed. The effectiveness of software implementation is achieved through the use of multi-threaded programming technology and processor power distribution using the built-in thread control libraries, as well as the use of modern methods and standard libraries.
- The software implementation for visualizing the learning process of the genetic algorithm and reinforcement learning to find the least resource-intensive path has been developed. The user interface is developed and tested for convenient user interaction with the software implementation of the algorithm.
- Genetic algorithm and reinforcement learning have been tested and compared. Based on a comparative analysis, recommendations on the choice of a learning algorithm are formulated.

References

1. Kenneth O.S.: Efficient Evolution of Neural Networks Through Complexification. Ph.D. Thesis. Department of Computer Sciences. The University of Texas at Austin (2004)
2. Wotkins C.H.: Learning from delayed rewards. Ph.D. Thesis. Cambridge (1989)
3. Kenneth O.S., Miikkulainen R.: Evolving Neural Networks Through Augmenting Topologies. Evolutionary Computation (2002)
4. Matthew E.T., Whiteson S., Stone P.: Comparing Evolutionary and Temporal Difference Methods in a Reinforcement Learning Domain. Proceedings of the Genetic and Evolutionary Computation Conference (2006)
5. Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein C.: Dijkstra's algorithm. Introduction to Algorithms (Second ed.). MIT Press and McGraw–Hill (2001)

6. Bellman R.: On a routing problem. Quarterly of Applied Mathematics, 16 (1958)
7. Zeng, W., Church, R. L.: Finding shortest paths on real road networks: the case for A*. International Journal of Geographical Information Science, 23:4, 531-543 (2009)
8. Floyd R. W.: Algorithm 97: Shortest Path. Communications of the ACM. 5, 6 (1962)
9. Johnson D. B.: Efficient algorithms for shortest paths in sparse networks. Journal of the ACM, 24 (1977)
10. Lee C. Y.: An Algorithm for Path Connections and Its Applications, IRE Transactions on Electronic Computers, EC-10 (1961)
11. Bellman R.: A Markovian Decision Process, Indiana Univ. Math. J. 6 No. 4, 679–684 (1957)