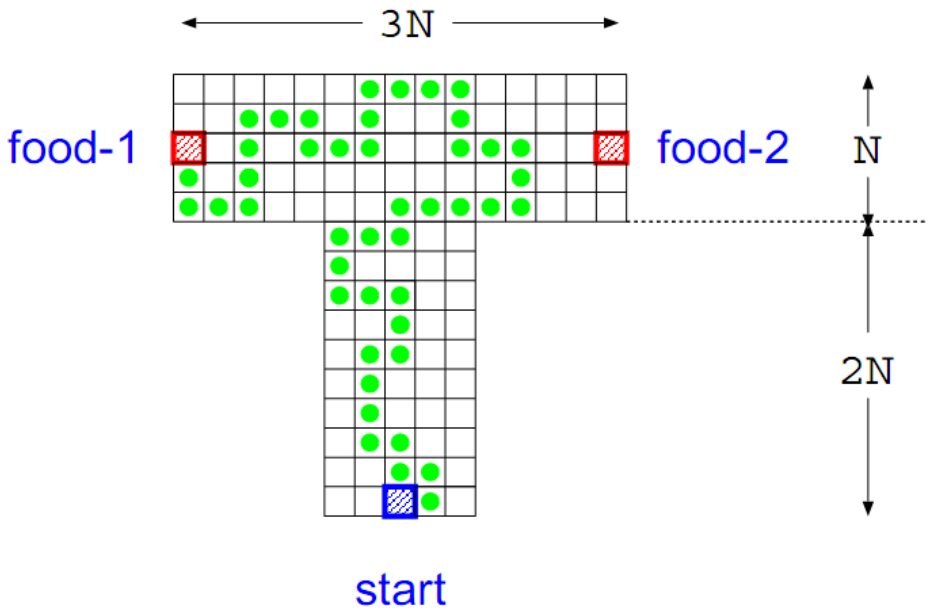
1. The T-maze Problem

In this project, we study machine learning techniques such as Genetic Algorithm[1], artificial Neural Network[2], Reinforcement Learning[3] by observing an artificial agent who travels a virtual world called a gridworld(you can see it in the picture 1.1.). The world has the point for the agent to start from, the point for the agent to aim as a goal, and the point for the agent to avoid. Usually we call “agent” who are to learn to behave intelligently, and here, agent’s aim is to reach the goal.

Agent can move only one cell at a time to the neighbor cell, that is, to up, down, right and left, unless the agent touches the border. When the agent touches the border, the action that makes the agent cross the border is not performed but it must remain stopped at the point waiting until the next action. For example, if the agent in Figure 1 is at (2, 6) and the action is to right, then agent remains at that point, and if the next action is to up, then it moves to (2, 7), or when the next action is left then agent moves to (3, 6).

Usually we call “agent” who are to learn to behave intelligently, and here, agent’s aim is to reach the goal. You may, however, replace the term an “agent” with a “dog,” and the “goal” with the “place where dog can find a sausage.” That is, a dog learns how it reaches the sausage efficiently, or with a shortest path.



Picture 1.1: A gridworld called T-maze and an example of the agent’s path

2. Designing neural networks and learning algorithms

2.1 Random walk

- Only good sosages are found at the food-site-1. No food at all at the food-site-2.
- Dog moves, from one step to the next, at random to up, down, right or left.
- If the movement is to cross the border of the gridworld, do not move, and wait for the next step. The same holds the followin all the experiment.
- Dog is allowed to move only limited number of steps, e.g., 100 steps for 15x15 gridworld.

2.2 Genetic Algorithm

The idea of the algorithm is based on Darwin's theory of biological evolution. The main idea: if you take two perfectly good solutions and somehow get out of them the new solution, we will have a high probability that the new solution will be better than the previous ones.

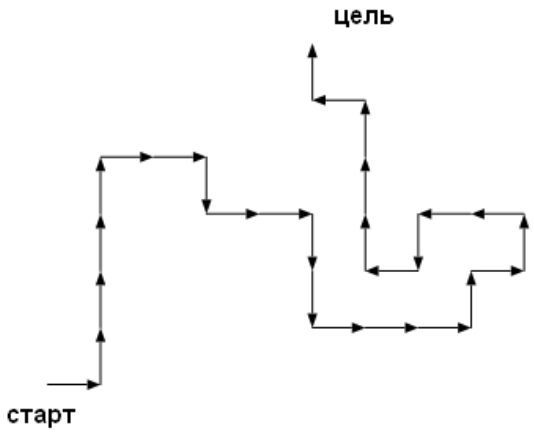
1) The maximum number of steps which the agent dies if it hasn’t reached the goal is k and it should be chosen depending on the size of the grid world (option N). Only good sosages are found at the food-site-1, no food at all at the food-site-2. The task is formalized in such a way that its decision could be coded as a vector ("genotype") genes, where each gene can be a weighting factor.

2)Creation of initial population, each dog in population is represented by it’s chromosome:

W = [w1, w2, ..., wi], i <= k
wi = 1 .. 4 (random initialization)

3) The agent tries to move in the direction (without going outside the world), which is set to wi:
if wi = 1, then moves up
if wi = 2, then moves down
if wi = 3, then moves to the right
if wi = 4, then moves to the left

(31111332332233131442411141)

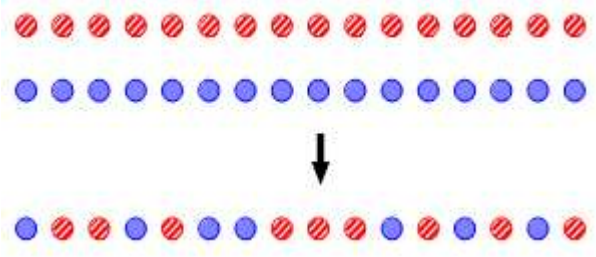


Picture 2.2.1. Trajectory of the dog depending on it’s genotype

4) Calculation of the fitness for each dog from the population . Fitness is equal to the Hamming distance to the positive goal.

5) We pick up two chromosomes at random from better half of the population (better chromosomes are more likely to be chosen) and create a child with the help of the "genetic operators": "crossover" and "mutation". By repeating this procedure we create next generation of dogs.

Picture 2.2.2.One-point-crossover



Picture 2.2.3. Uniform-crossover

Picture 2.2.4. An example of mutation

6) Repeat steps 3-5 several times (generations) until a stopping criterion of the algorithm is executed. Such a criterion might be:

- finding a global or suboptimal solutions
- exhaustion of the number of generations allotted for evolution
- exhaustion of the time allotted for evolution

2.3 Reinforcement Learning

In Reinforcement Learning (RL), an agent takes one state at a time chosen from predefined all possible states. In each of those state, also all possible actions are given one by one with a probability of how likely the action will be made. State is the cell in which the agent is located. So we have 5 M^2 different states. The possible actions is either up, down, right and left.

To choose the action in one state, we employ what we call E-greedy strategy. That is, we choose an action at random with probability E (a pre-fixed small value such as 0.1) and the action with the highest value in the Q-table with probability 1 − E.

One method of renewing the policy table is called Q-learning. Assume now the state of the agent at time t is s_t and the action policy table gives is a_t.

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \times \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t)),$$

(2.3.1)

where

$\alpha = 0...1$ - learning ratio

$\gamma = 0...1$ - discount rate

$\max_a Q(s_{t+1}, a_t)$ - maximum future value for the best action in the current state

Reinforcement learning algorithm:

1) set all the values in q-table to 0, set the desired values of the coefficients α и γ and establish the fact of the existence of the ability of an agent to find out information about the distance to the target(to know fitness).

2) choose an action at random with probability E (a pre-fixed small value such as 0.1) and the action with the highest value in the Q-table with probability 1 − E(if all the actions in Q-table have equal values, each of them will have

probability 0.25)

3) For the current state and chosen action change the Q-table according to the formula(2.3.1).

Use following rules for calculating the reward for the selected action:

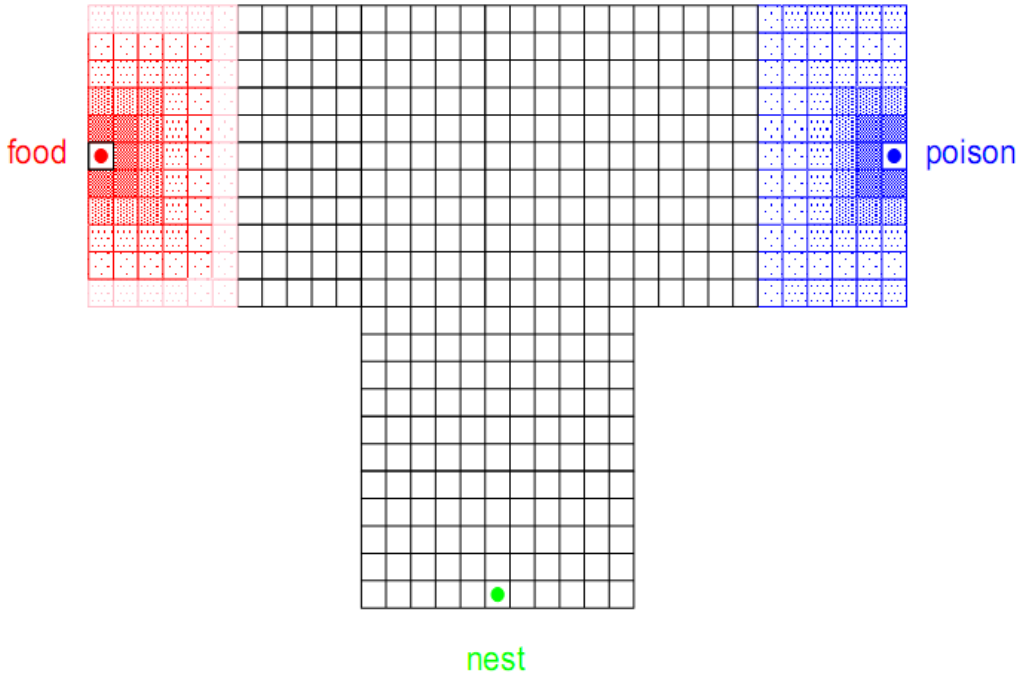
- If the agent will reach a good aim at time t +1, the reward is equal 500, but if it will be a bad aim, the reward is -500
- If agent tries to cross the world's border, give the reward -10.
- If the agent moves to the next empty cell and he has no opportunity to receive information about the distance to the target, then the award is -1, but when such information is available, he gets the award 3 when he will appear closer to the good aim or -3 when in other case.

4) The agent performs a selected on the second step action, but he mustn't cross the border of the gridworld.

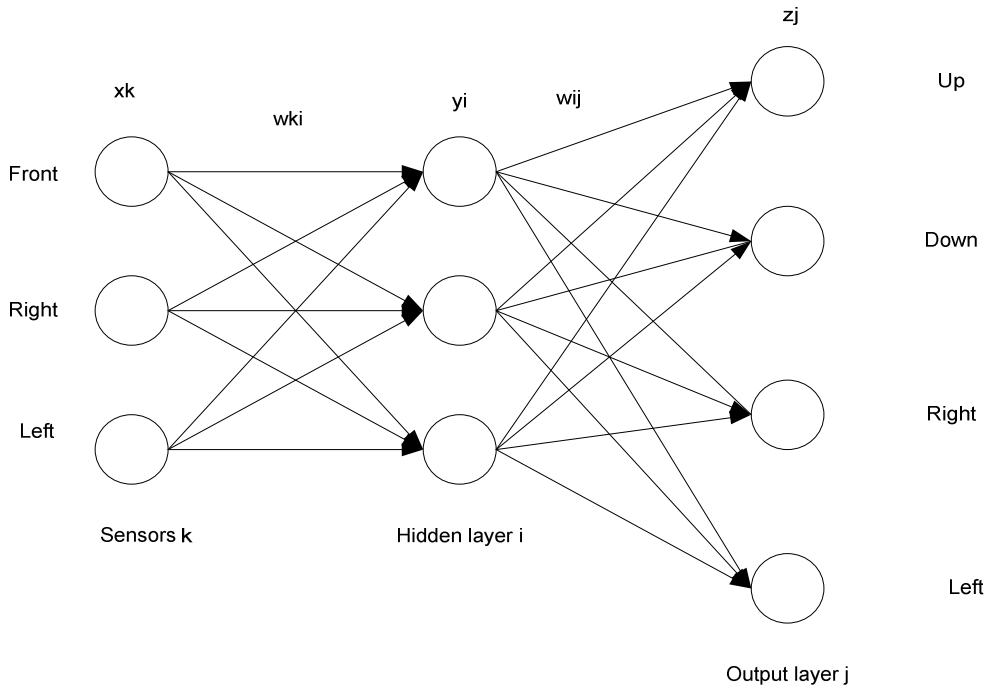
5) Steps 2 - 4 are repeated until the agent reaches a good aim.

2.4 Neural Network

In this experiment, we have good sausages at site-1, and potatoes, or very old sausages (poison) at site-2. Neural network has three sensor inputs which detect smell of its front-left, front, front-right. And the number of output is four and follow the winner-take-all rule, for example. Each output corresponds to up, down, right and left. Also neural net work has a hidden layer with three neurons. In both of the good smell and bad smell, the closer to the food the stronger the smell(picture 2.4.1). The architecture of the neural network with one hidden layer of three neurons is shown in picture 2.4.2.



Picture 2.4.1. Each food has a smell, the closer to the food the stronger the smell



Picture 2.4.2. The architecture of the neural network with one hidden layer.

21 weights and 7 thresholds for neurons in the hidden and output layers should be adjusted by genetic algorithm. Robot's behaviour is based on the values of the sensors(x_k).

Output activity of neurons is calculated by formula:

$$yi = F(\sum_k \omega_{ki} y_k - T_i) \quad (2.4.1)$$

, where j characterizes neurons of the next layer to i-th layer.

Let's use sigmoid activation function for neurons in the hidden layer:

$$F(x) = \frac{1}{1+e^{-x}} \quad (2.4.2)$$

Let's use sigmoid activation function for neurons in the output layer:

$$F(x) = x \quad (2.4.3)$$

Solution is based on the values of the neurons in the output layer according to the rule "winner takes all."

Learning algorithm for network with one hidden layer(sigmoid activation function) using the genetic algorithm:

1) The maximum number of steps k, after which the agent dies if he hasn't reached the goal should be chosen depending on the size of the grid world (option N).

2) generate random chromosome (it includes weights and thresholds) for every agent in the initial population:

W = [w1, w2, ..., w21]

T = [T1, T2, ..., T7]

wi = 0 ... 1, Ti = 0 ... 1

3) Calculate the values of neurons in hidden and output layers.

Hidden layer:

$$yi = F(\sum_k \omega_{ki} x_k - T_i)$$

Output layer:

$$y_j = \sum_i \omega_{ij} y_i - T_j$$

3) The agent tries to move in the direction (but he mustn't cross the world's border), which is determined by the winner neuron("winner takes all") in the output layer:

• If the first neuron won, the agent moves up

• If the second neuron won, the agent moves down

• If the third neuron won, the agent moves to the right

• If the fourth neuron, the agent moves to the left

4) Calculate the fitness for each agent from the generation . Fitness is equal to the

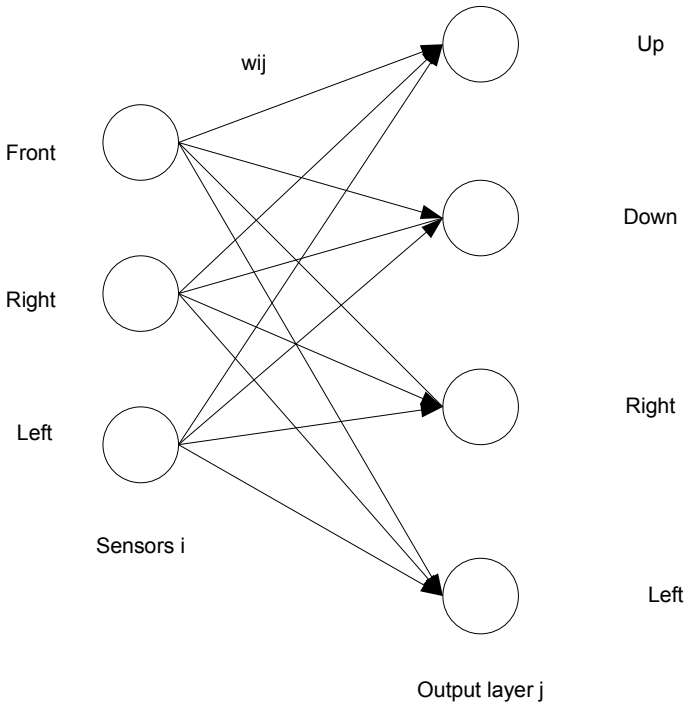
Hamming distance to the positive goal.

5) We pick up two chromosomes at random from better half of the population (better chromosomes are more likely to be chosen) and create a child with the help of the "genetic operators": "crossover" and "mutation". By repeating this procedure we create next generation of dogs.

6) As a result of genetic operators are obtained by new individuals and new results. For them also calculated fitness value, and then are selected (selection) the best solutions in the next generation.

7) Repeat steps 3-6 several times (generations).

For the sake of the experiment, let’s also explore a single-layer perceptron trained by genetic algorithm and try to compare the results. The architecture of the single-layer perceptron is shown in picture 2.4.3.



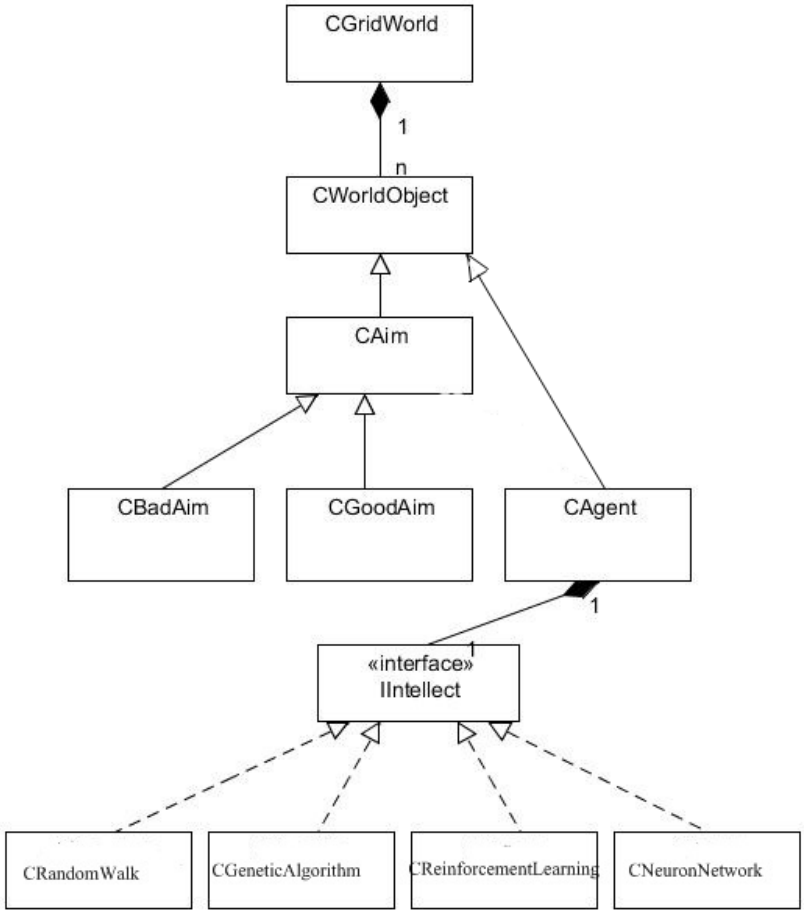
Picture 2.4.3. Architecture of single-layer neural network

In this case we have to configure 12 weights and 4 thresholds. Learning algorithm is practically the same as for a multilayer perceptron(without hidden layer), the values of neural elements in the output layer are calculated by formula (2.4.1). Activation function is linear.

3. Development of software modules

The key class is CGridWorld – it describes the gridworld, which can be filled with various objects of class CWorldObject. Objects can be of 2 types: agents(CAgent) and aims(CAim), also there are 2 types of aims: positive (CGoodAim) and negative (CBadAim). Agent moves in a grid world, according to the rules laid down at creation in his “brain”(class IIntellect). There are 4 different implementations of the class IIntellect: CRandomWalk, CGeneticAlgrithm, CReinforcementLearning, CNeuronNetwork. All the experiments described in the class Test.

Class hierarchy is shown in the picture 3.1.



Picture 3.1. Class Hierarchy

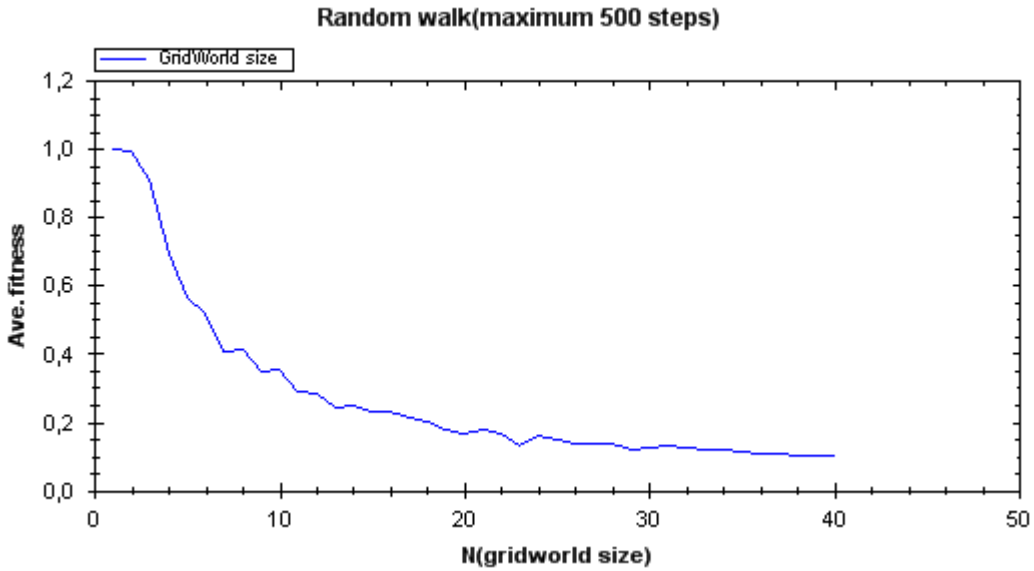
4. Results of testing and comparative analysis

4.1 Testing

During testing, widely used feature adaptation (0 - the agent is at a maximum distance from the target, 1 - the agent found the goal)

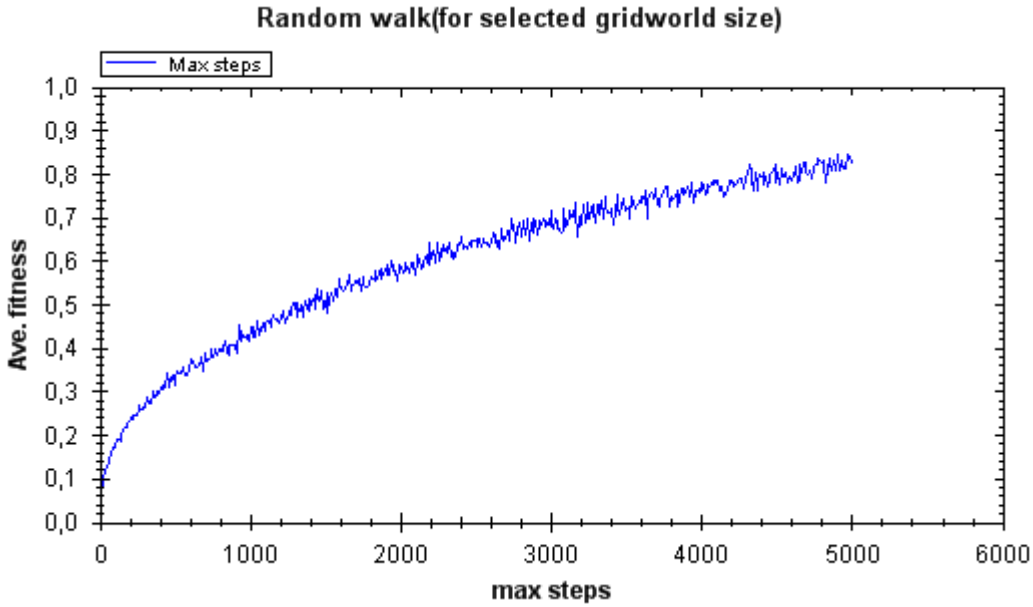
Random walk

Let’s investigate the dependence of fitness on size of the grid world (maximum number of steps is 500).

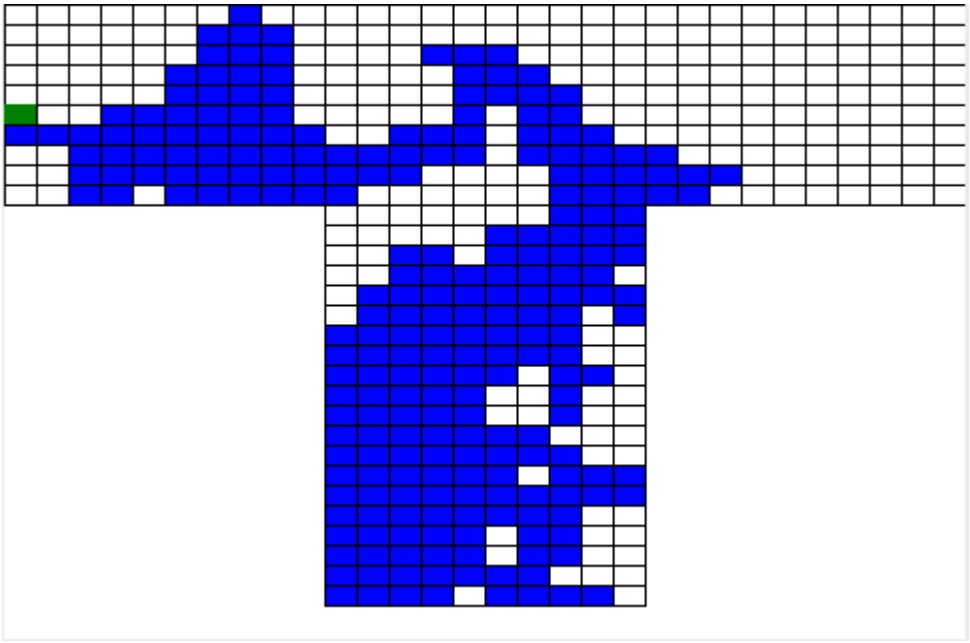


Picture 4.1.1. The dependence of fitness on the size of the grid world (maximum number of steps is 500)

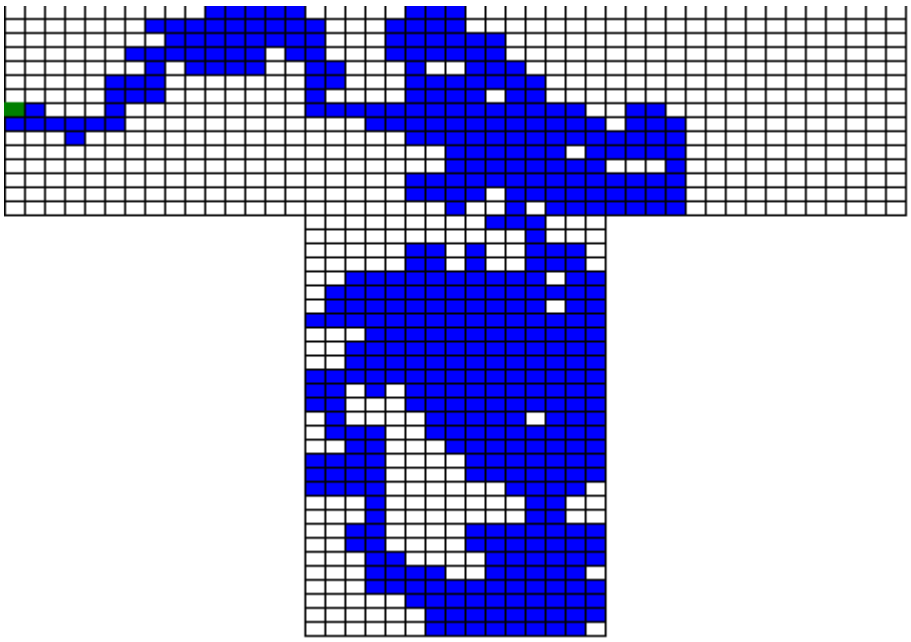
Let’s investigate the dependence of fitness on maximum number of steps(N = 10).



Picture 4.1.2. The dependence of fitness on maximum number of steps(N = 10)



Picture 4.1.3. An example of a good agent's trajectory for N = 10



Picture 4.1.4. An example of a good agent's trajectory for N = 15

Results

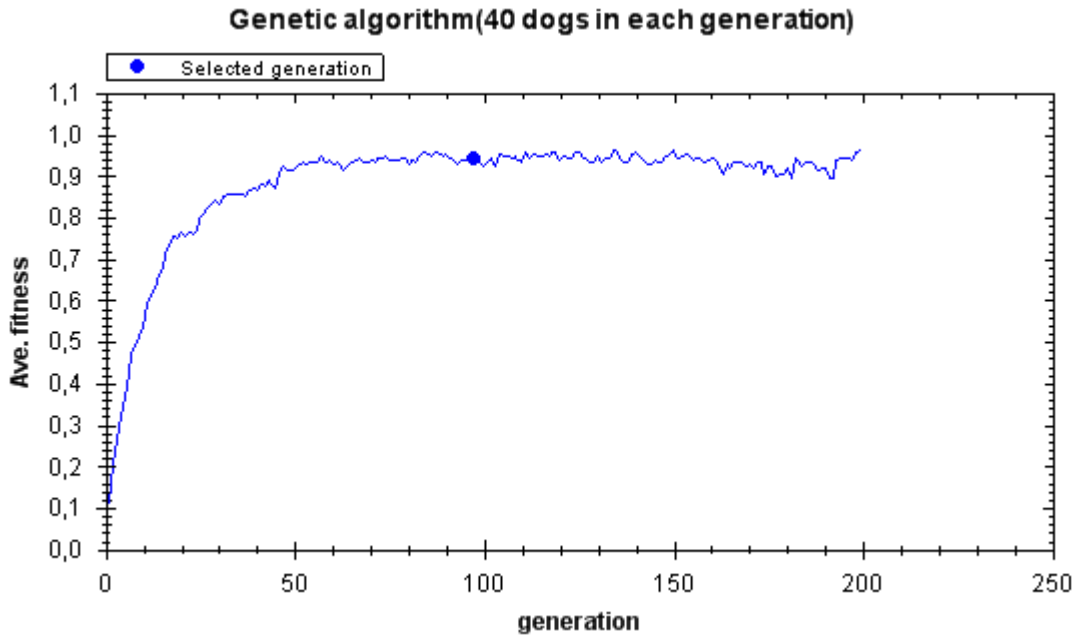
- Increase of the size of the gridworld decreases the agents' fitness (for the constant maximum number of steps).
- Increase of maximum number of steps increases the agents' fitness (for the constant gridworld size).
- Random walk algorithm gives at most bad results for the problem.

Genetic algorithm

Let's investigate the dependence of average population's fitness on generation number (40 agents in every population). Experiments will be conducted for different maximum number of steps(maximum number of steps determines number of genes in the chromosome).

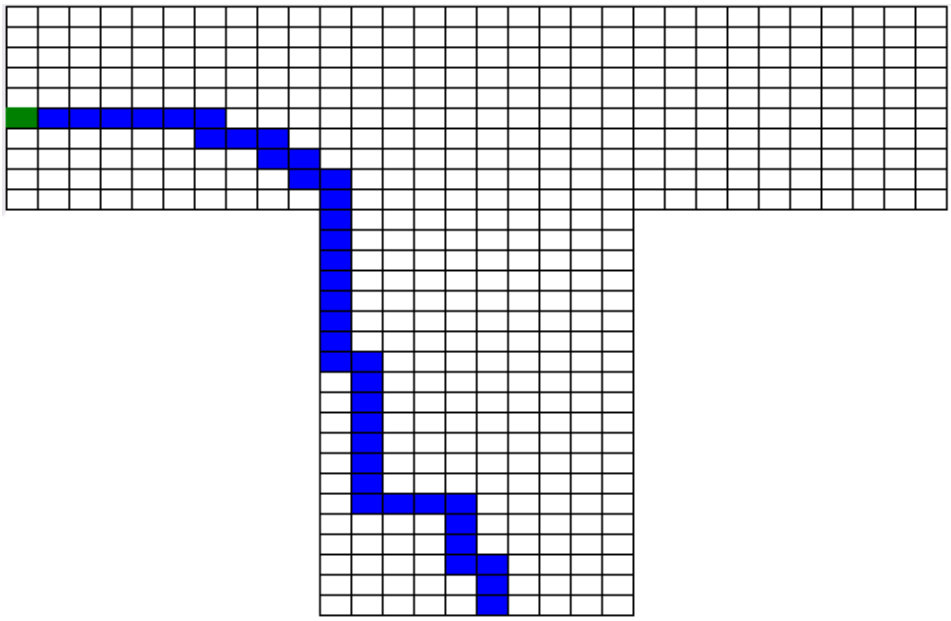
1) N = 10

1.1) Maximum number of steps is 40(4 * N = 40 is minimum required number of steps to achieve the goal)



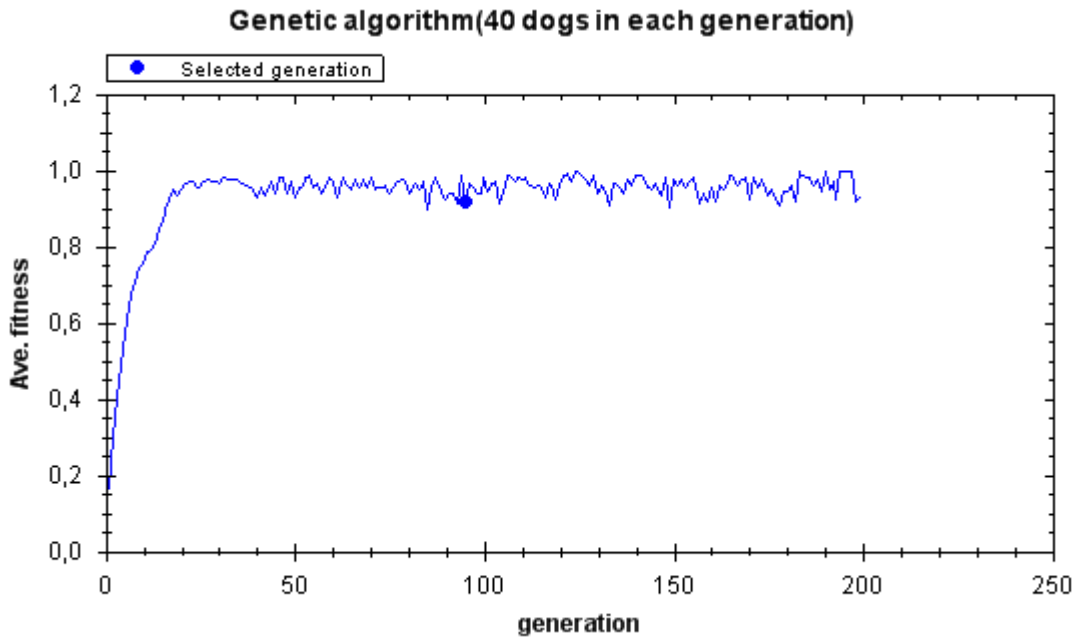
Picture 4.1.5. The dependence of the average fitness on the generation number(maximum 40 steps, N = 10)

As we can see already somewhere in the 50 generation the average fitness of the whole population tends to 1 (nevertheless there is some dependence on the initialization of chromosomes). Let's observe the trajectory of one of the best agents of the 100 generation.



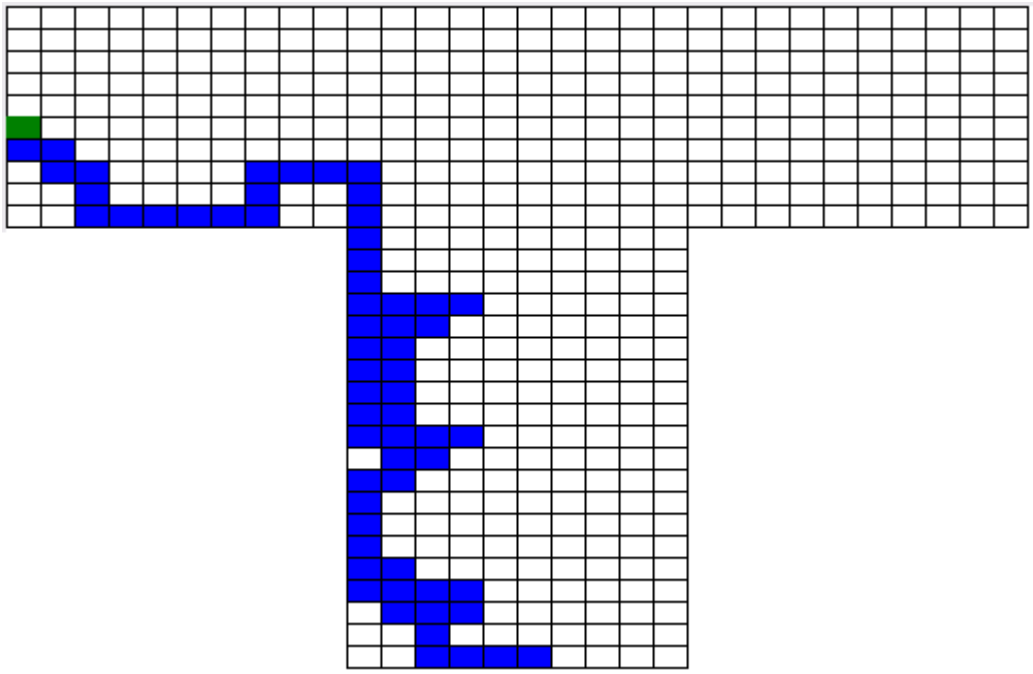
Picture 4.1.6. An example of the best agent's trajectory of the 100 generation(maximum 40 steps, N = 10)

1.2) Maximum number of steps is 100**(** N^2 **)**



Picture 4.1.7. The dependence of the average fitness on the generation number(maximum 100 steps, N = 10)

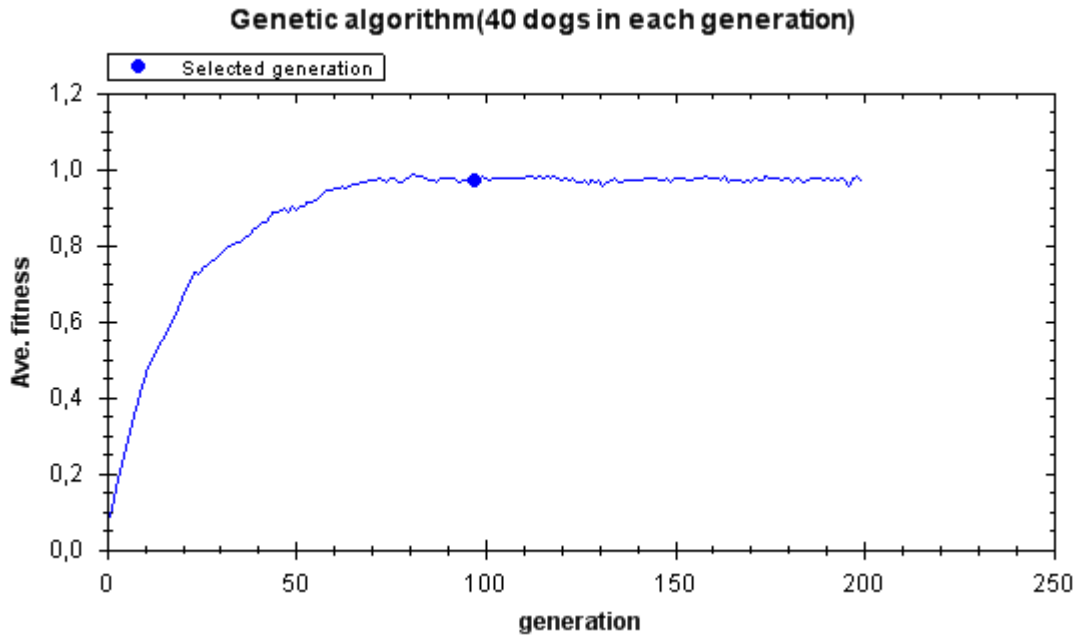
As we can see already somewhere in the 20 generation the average fitness of the whole population tends to 1 (nevertheless there is some dependence on the initialization of chromosomes). Let’s observe the trajectory of one of the best agents of the 100 generation.



Picture 4.1.8. An example of the best agent's trajectory of the 100 generation(maximum 100 steps, N = 10)

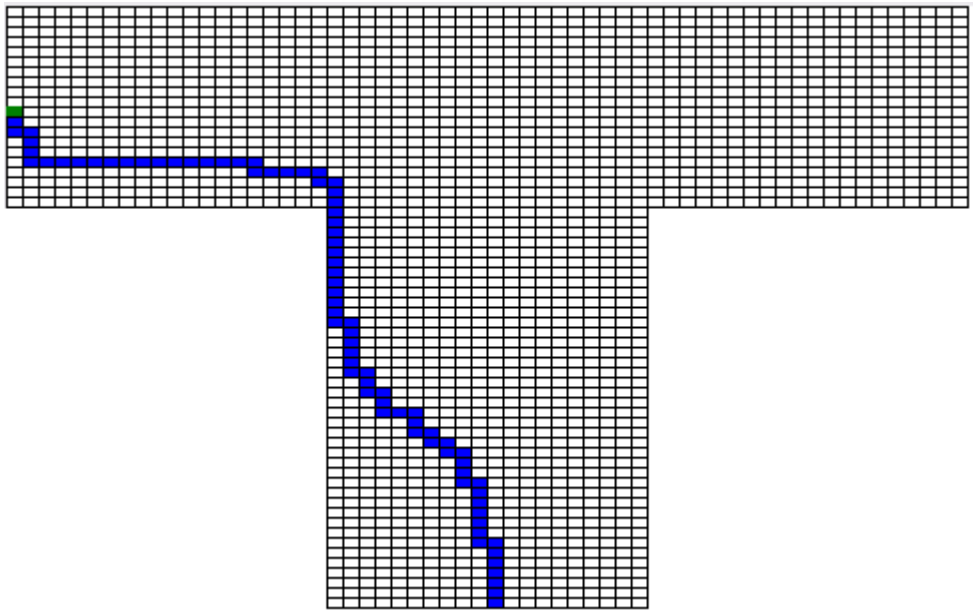
2) N = 20

2.1) Maximum number of steps is 80(4 * N = 80 is minimum required number of steps to achieve the goal)



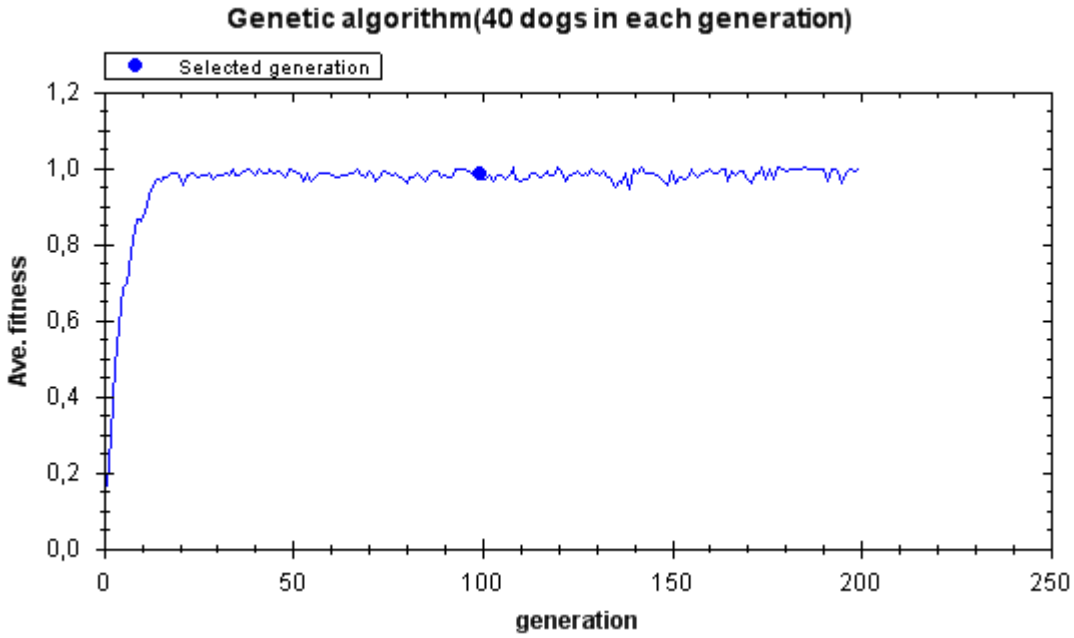
Picture 4.1.9. The dependence of the average fitness on the generation number(maximum 80 steps, N = 20)

As we can see already somewhere in the 75 generation the average fitness of the whole population tends to 1 (nevertheless there is some dependence on the initialization of chromosomes). Let's observe the trajectory of one of the best agents of the 100 generation.



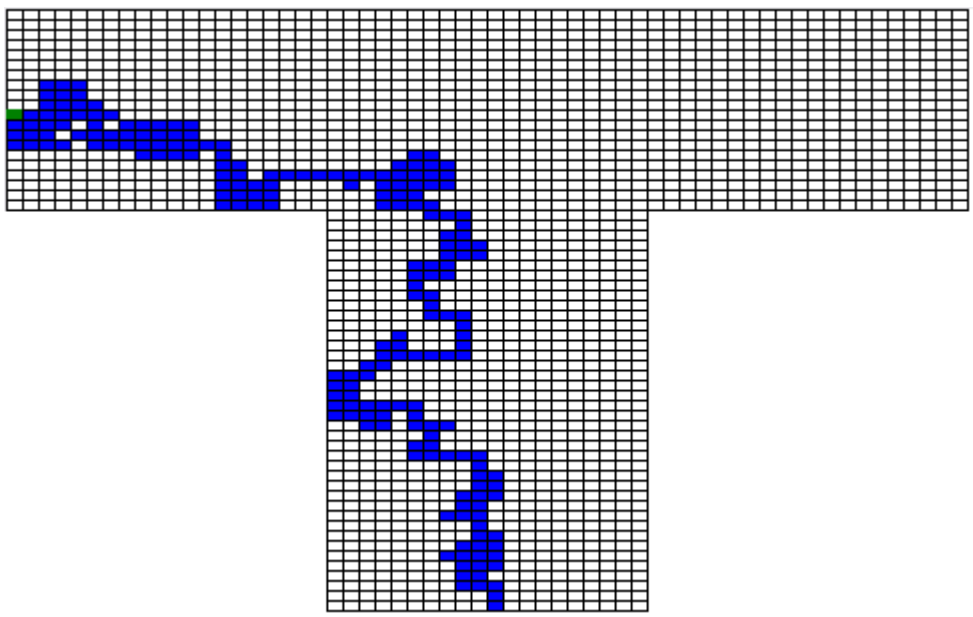
Picture 4.1.10. An example of the best agent's trajectory of the 100 generation(maximum 80 steps, N = 20)

2.2) Maximum number of steps is 400(N^2)



Picture 4.1.11. The dependence of the average fitness on the generation number(maximum 40 steps, N = 20)

As we can see already somewhere in the 20 generation the average fitness of the whole population tends to 1 (nevertheless there is some dependence on the initialization of chromosomes). Let's observe the trajectory of one of the best agents of the 100 generation.



Picture 4.1.12. An example of the best agent's trajectory of the 100 generation(maximum 400 steps, N = 20)

Results

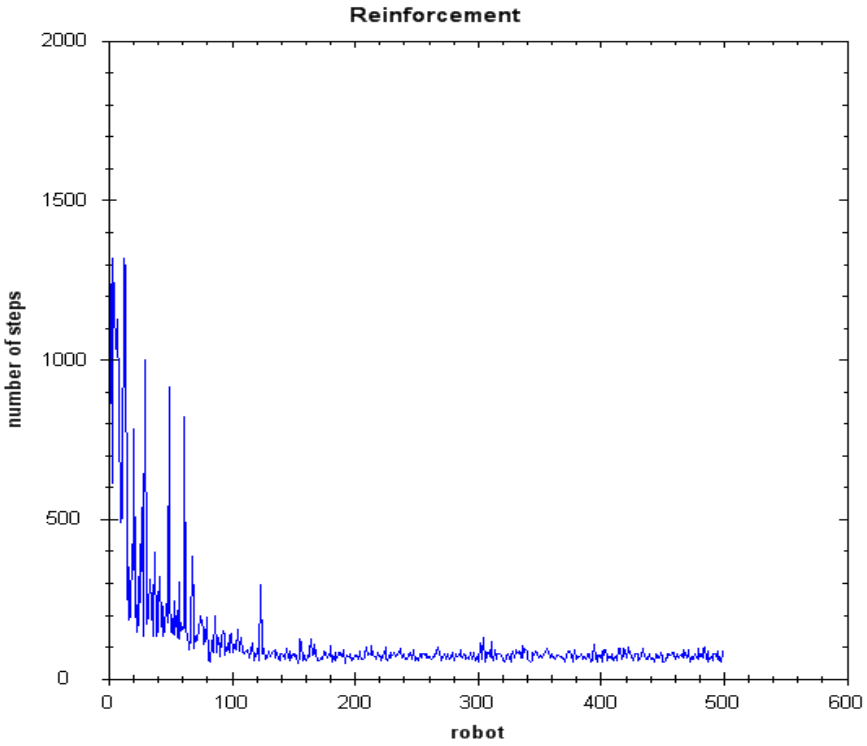
- The genetic algorithm gives the best results with a minimal count of maximum number of steps (and correspondingly with the minimum required length of chromosomes). However, with bigger maximum number of steps fitness increase is faster.
- One-point-crossover and uniform-crossover gives in average similar results.
- The algorithm is able to provide for different sizes of the gridworld in average equally good results. At the same time, on the output we can take a family of different optimal paths.

Reinforcement learning

Let’s investigate the dependence of agent’s fitness on number of attempts to reach the goal.

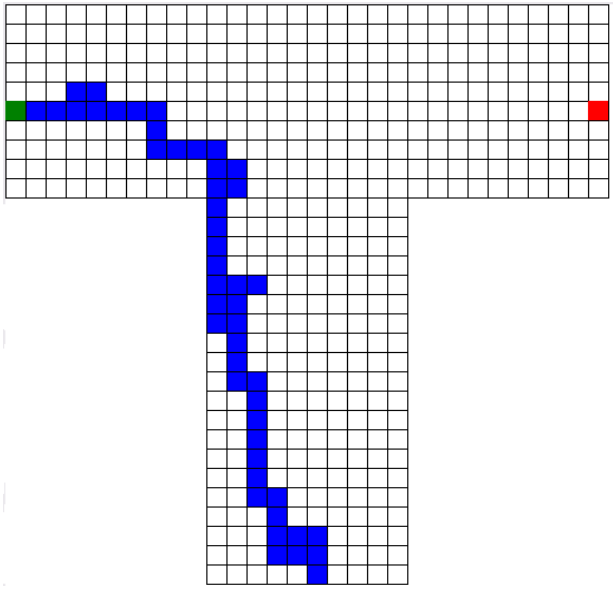
1) N = 10

1.1) agent has no information about the distance to the good aim

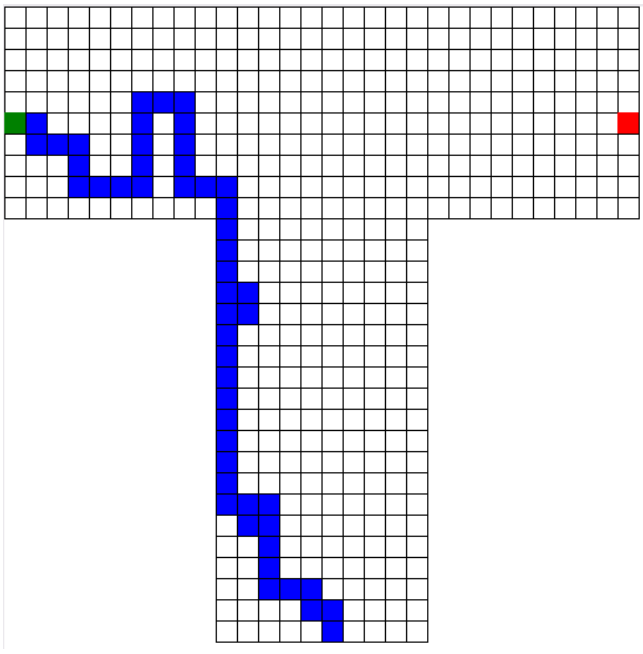


Picture 4.1.13. The dependence of agent’s fitness on number of attempts to reach the goal(N = 10, without information about the distance to the good aim)

Examples of robot's trajectories on the 200-th and 300-th attempts can be seen below in the pictures.

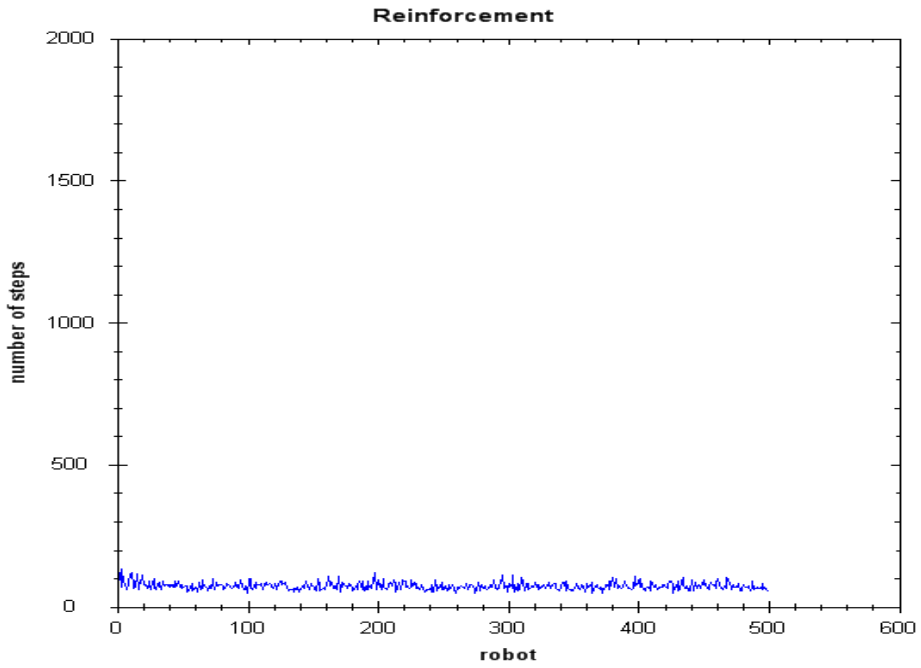


Picture 4.1.14. An example of the agent's trajectory on the 200-th attempt(N = 10, without information about the distance to the good aim)



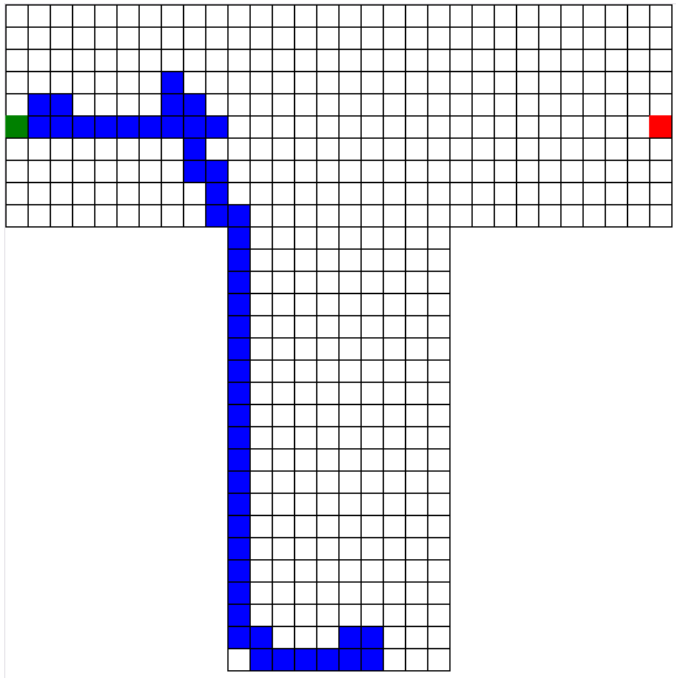
Picture 4.1.15. An example of the agent's trajectory on the 300-th attempt(N = 10, without information about the distance to the good aim)

1.2) agent has information about the distance to the good aim

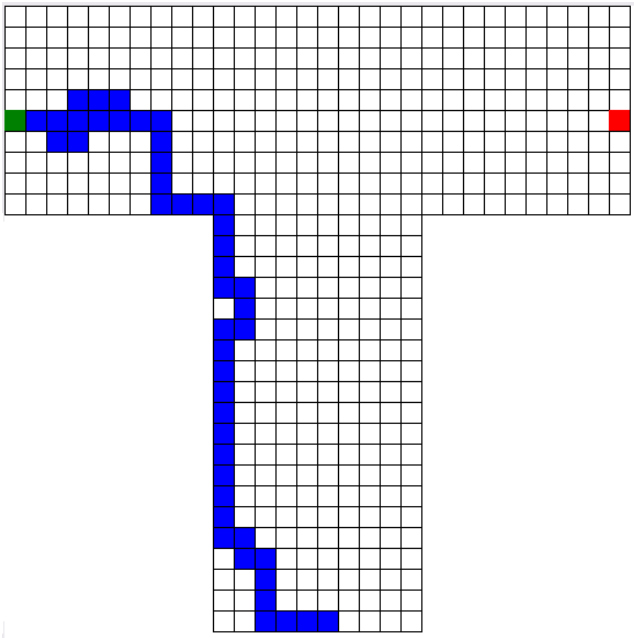


Picture 4.1.13. The dependence of agent’s fitness on number of attempts to reach the goal(N = 10, with information about the distance to the good aim)

Examples of robot's trajectories on the 200-th and 300-th attempts can be seen below in the pictures.



Picture 4.1.14. An example of the agent's trajectory on the 200-th attempt(N = 10, with information about the distance to the good aim)

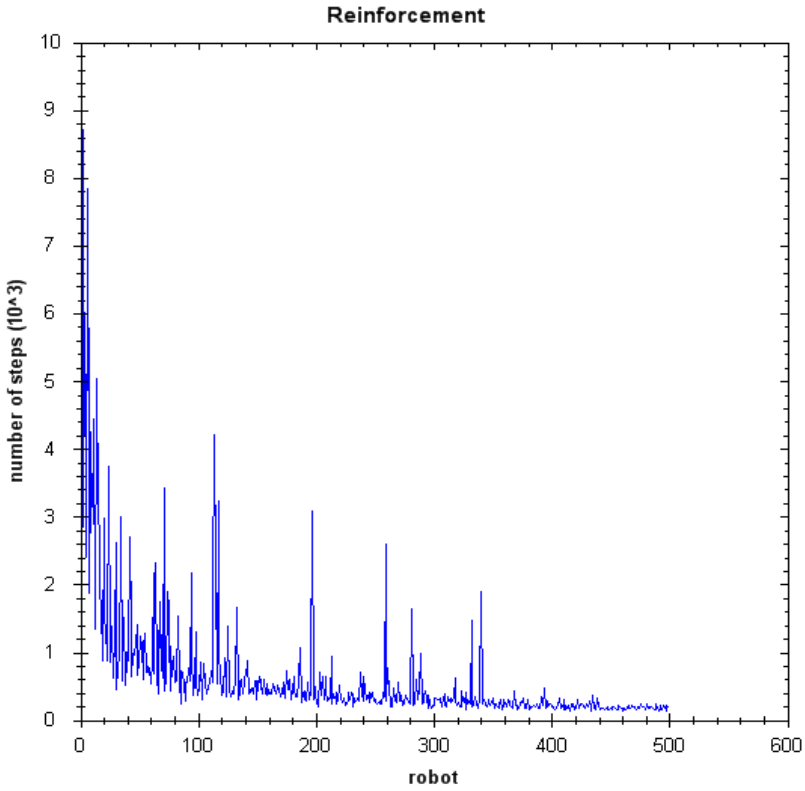


Picture 4.1.15. An example of the agent's trajectory on the 300-th attempt(N = 10, with information about the distance to the good aim)

As we can see in the pictures, ability to recognize the distance to the target slightly accelerates the learning process and improve results for small gridworlds.

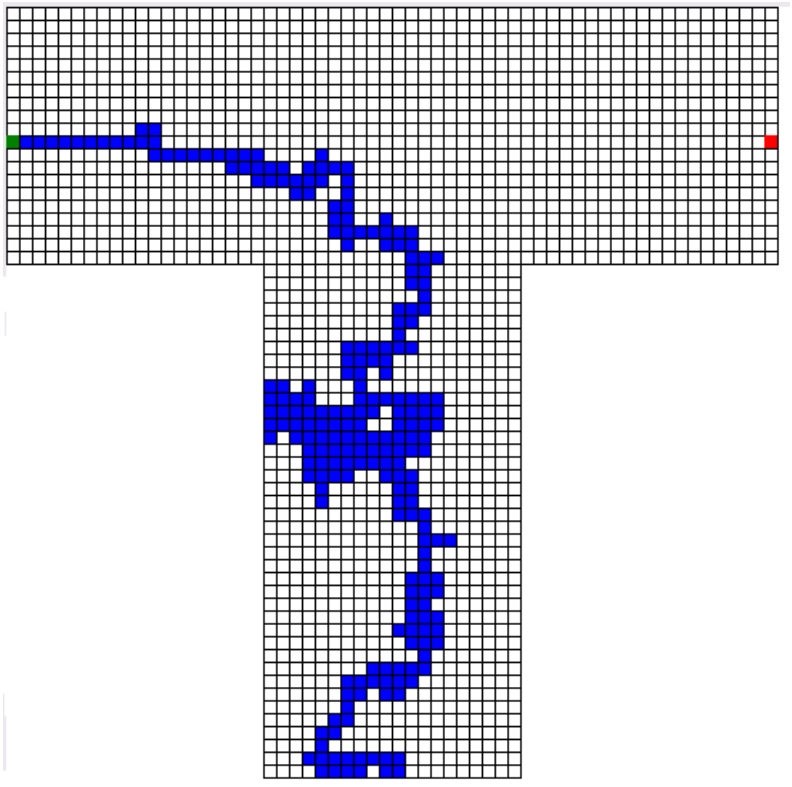
2) N = 20

2.1) agent has no information about the distance to the good aim

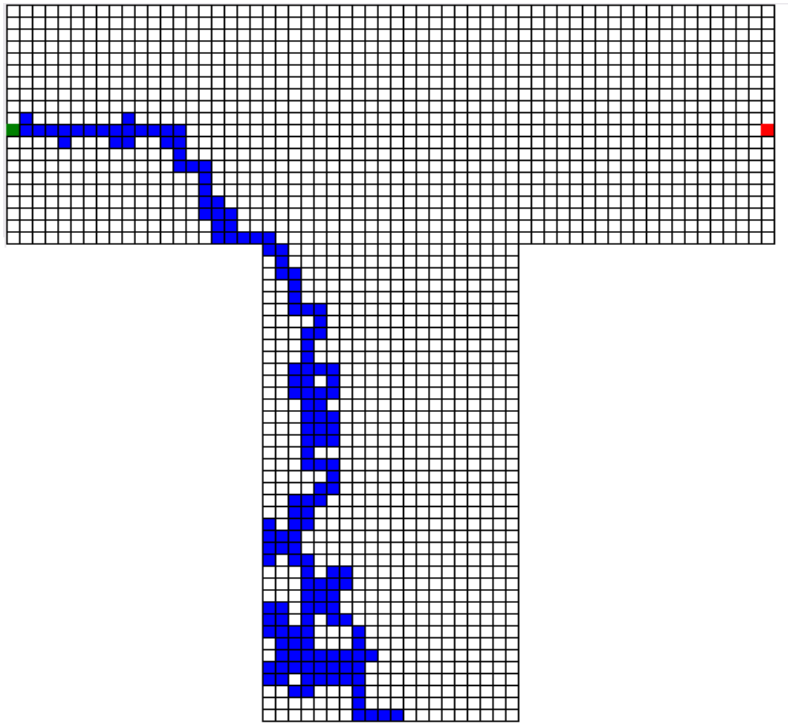


Picture 4.1.16. The dependence of agent's fitness on number of attempts to reach the goal(N = 20, without information about the distance to the good aim)

Examples of robot's trajectories on the 100-th and 300-th attempts can be seen below in the pictures.

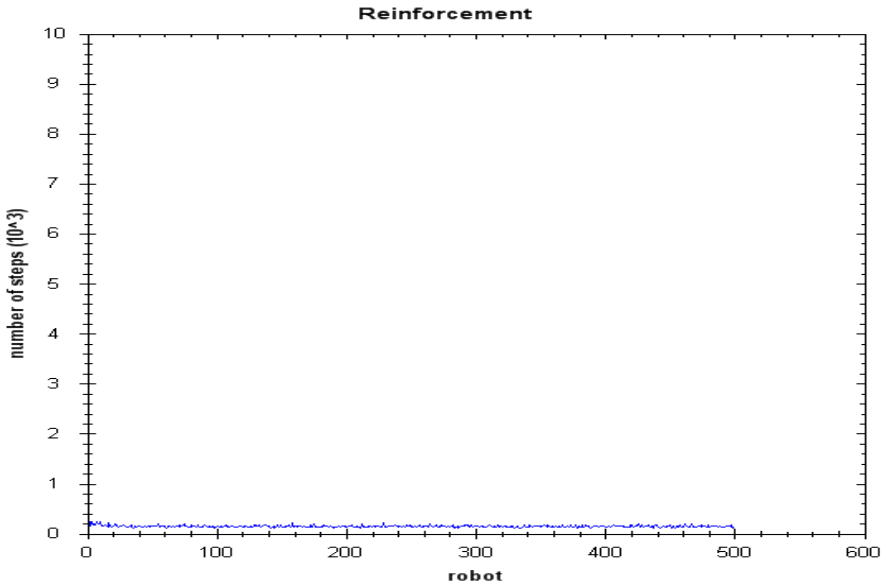


Picture 4.1.17. An example of the agent's trajectory on the 100-th attempt(N = 20, without information about the distance to the good aim)

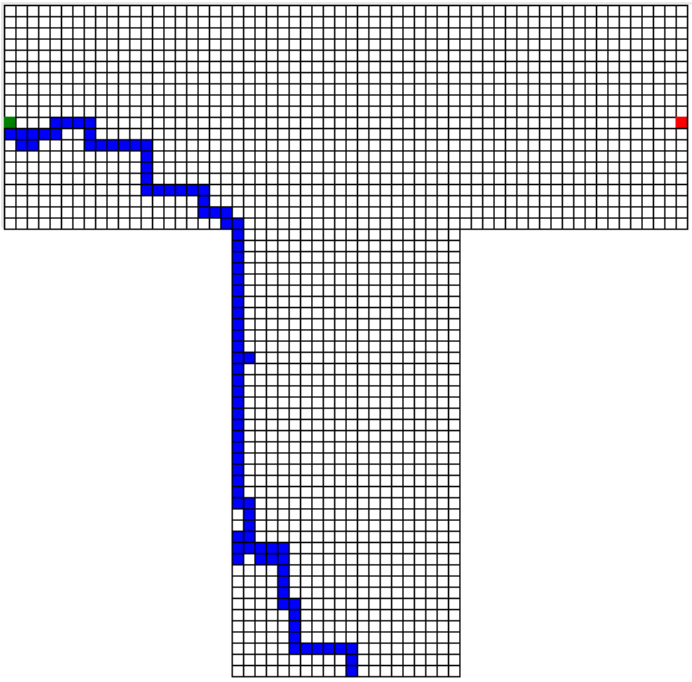


Picture 4.1.18. An example of the agent's trajectory on the 300-th attempt(N = 20, without information about the distance to the good aim)

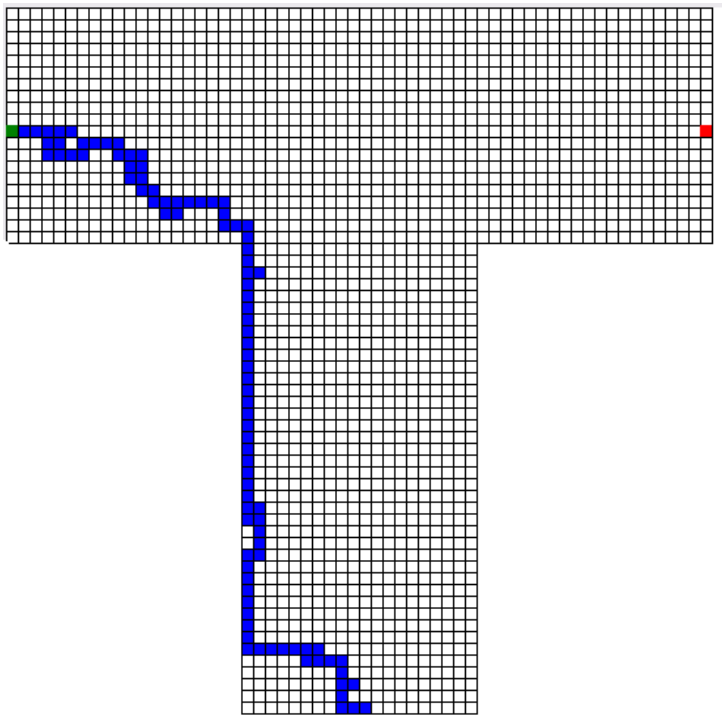
2.2) agent has information about the distance to the good aim



Picture 4.1.19. The dependence of agent’s fitness on number of attempts to reach the goal(N = 20, with information about the distance to the good aim)



Picture 4.1.20. An example of the agent's trajectory on the 100-th attempt(N = 20, with information about the distance to the good aim)



Picture 4.1.21. An example of the agent's trajectory on the 300-th attempt(N = 20, with information about the distance to the good aim)

As we can see in the pictures, ability to recognize the distance to the target greatly accelerates the learning process and improve results for big gridworlds.

Results

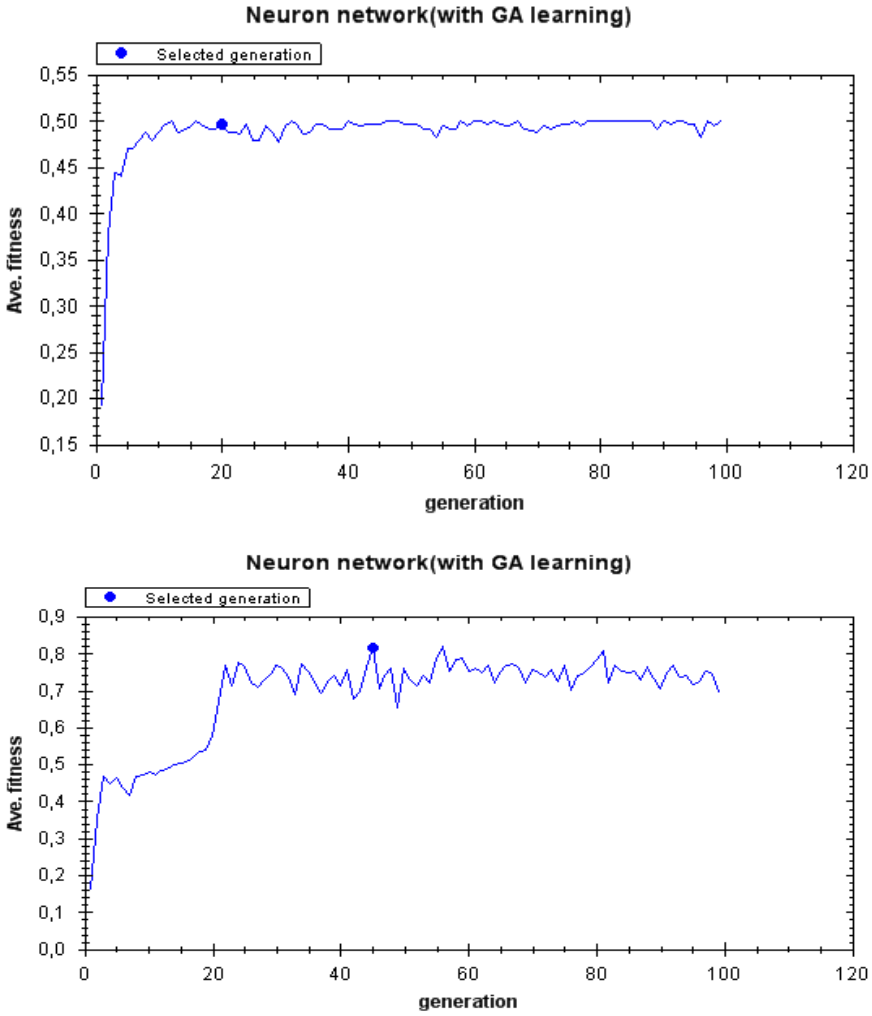
- The algorithm can give a family of different good(but not always optimal) paths.
- Ability to recognize the distance to the target is worth for using for big gridworlds, because it greatly accelerates the learning process and improve results. Ability to recognize the distance to the target isn't worth for using for small gridworlds, because it is probably not yield the desired improvement of results commensurate with the cost of necessary equipment for determining a distance.

Neural network

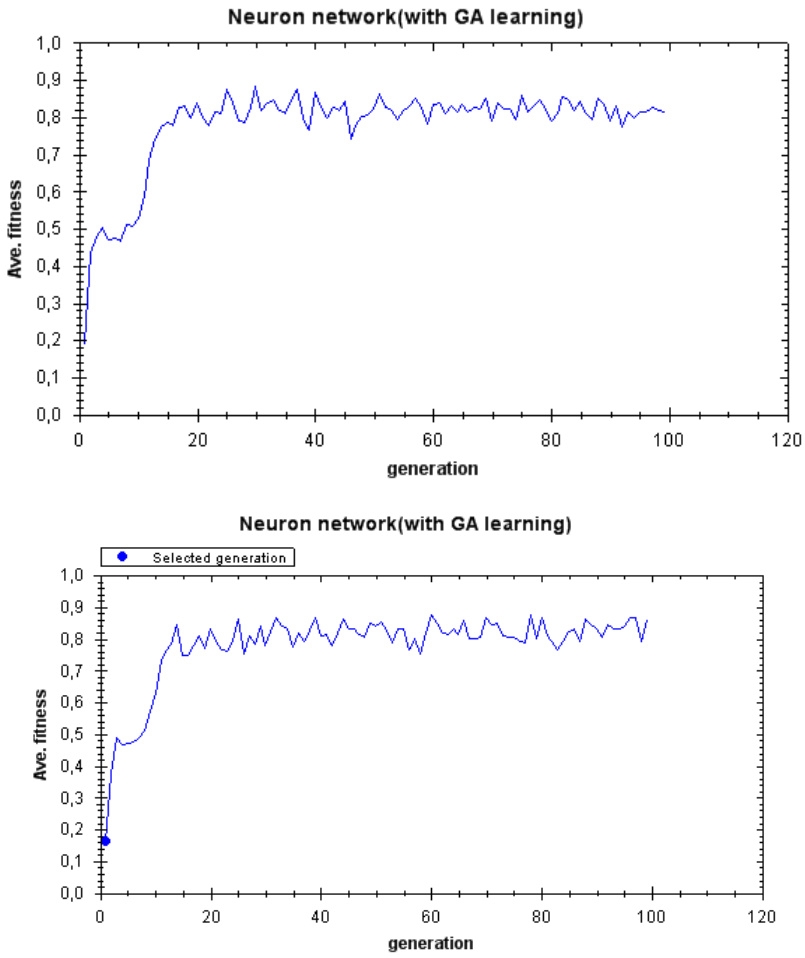
Let’s investigate the dependence of average population’s fitness on generation number (100 agents in every population).

1) One-hidden layer neural network with a sigmoid activation function.

During the study it was found that using multilayer neural network with genetic algorithm gives worse results than using single-layer network. Such a network is more difficult to study and the final results strongly depend on the initialization of the chromosomes (see in the picture 4.1.22), so in the future we will consider single-layer perceptron learning. Also it was found that the maximum number of steps does not affect the agents’ results(see in the picture 4.1.23) and for simplicity in future we well use the minimum number of steps 4 * N.



Picture 4.1.22. The dependence of the average fitness on the generation number for different initialization(N = 20)

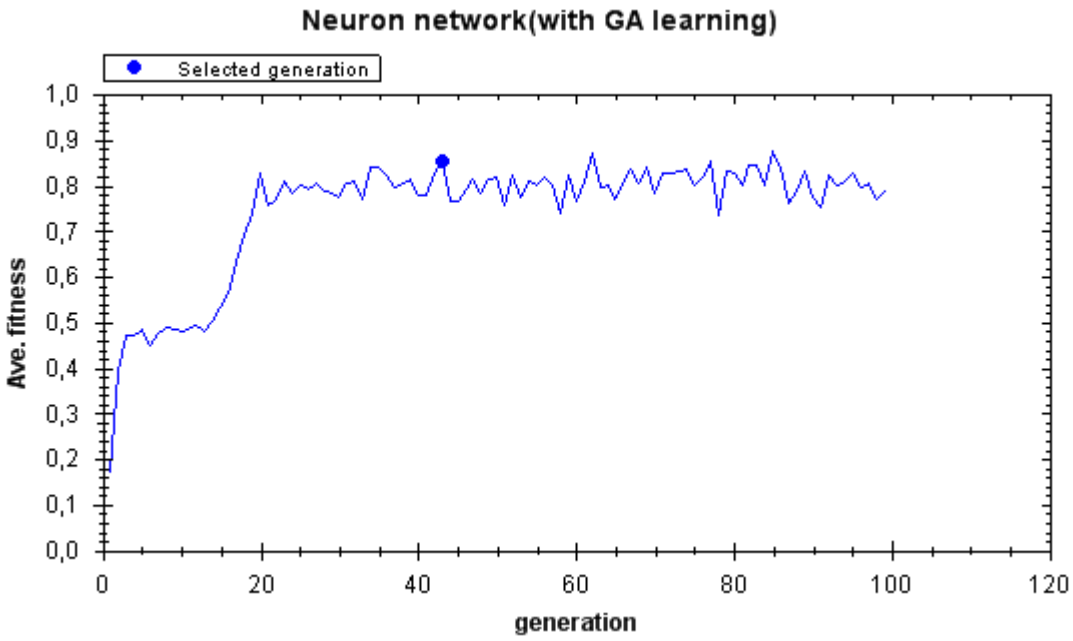


Picture 4.1.23. The dependence of the average fitness on the generation number (maximum number of steps in the first picture is $4*N$, in the second picture it is N^2 ,N = 20)

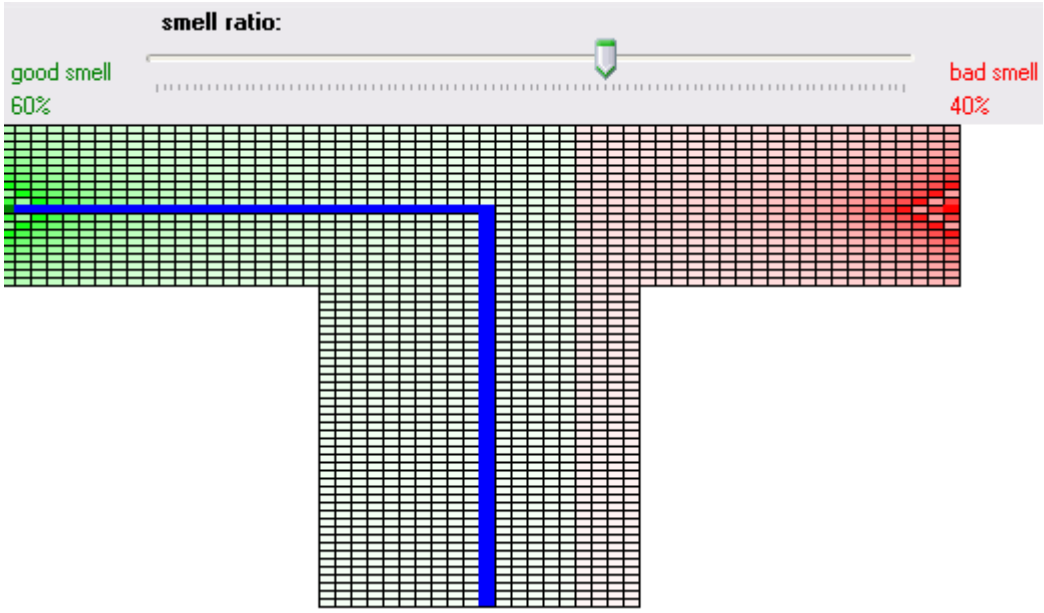
2) One-layer perceptron

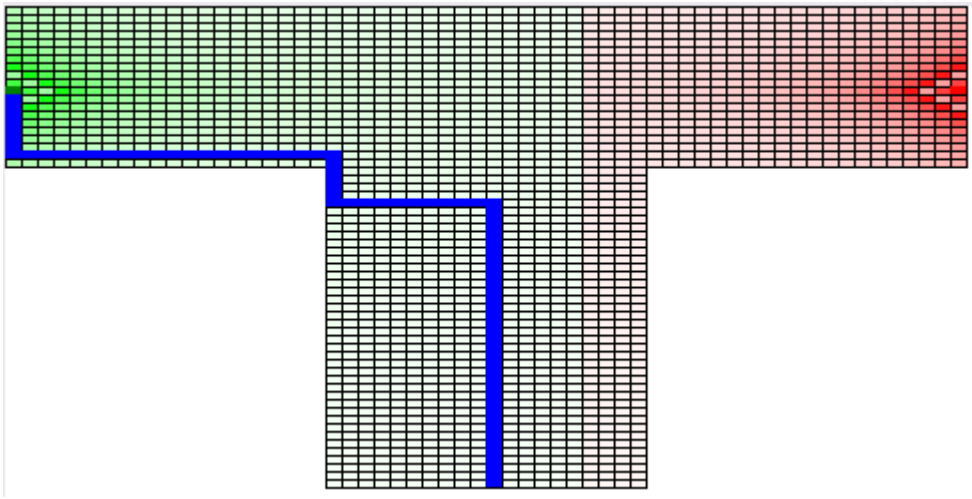
Also it was found that the maximum number of steps does not affect the agents' results. So we restrict the study of the gridworld with N = 20.

2.1) The ratio of good to bad is 60 % to 40 %.



Picture 4.1.22. The dependence of the average fitness on the generation number(N = 20, 60 % of good smell)

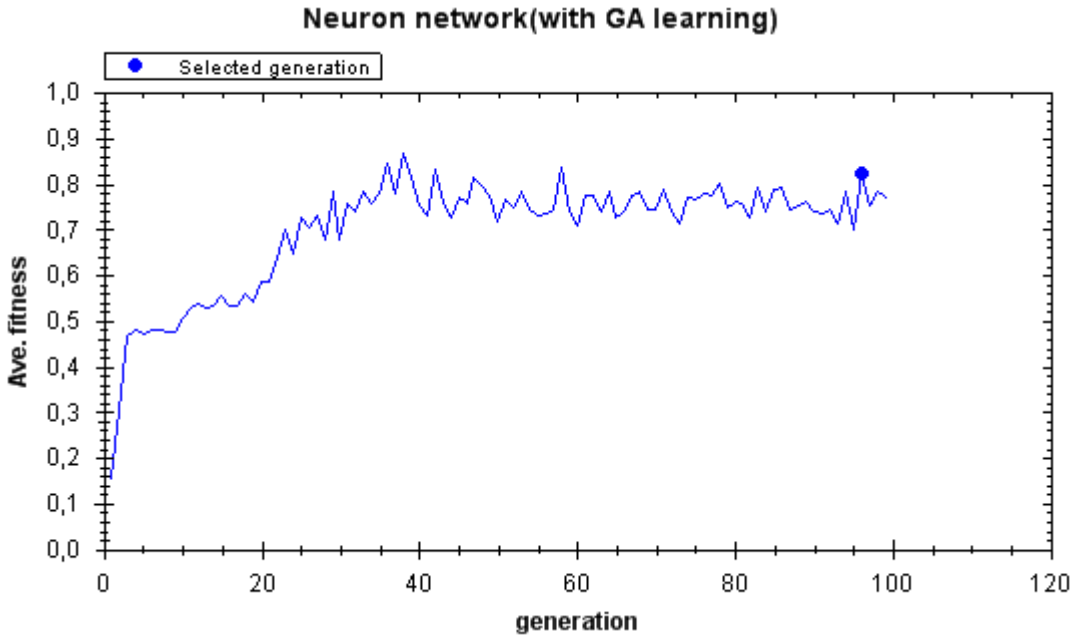




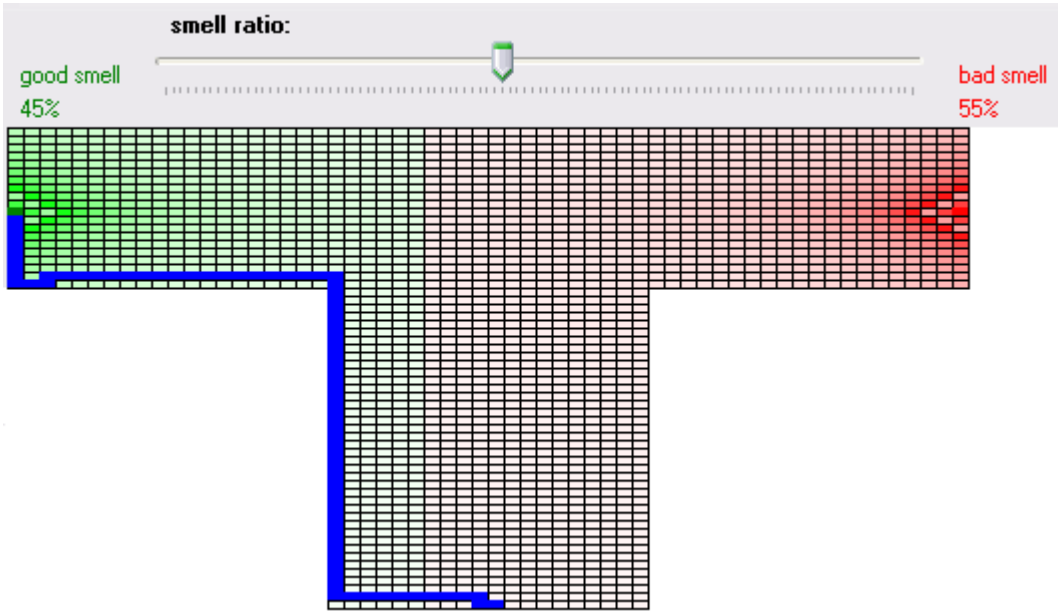
Picture 4.1.23. Examples of the agent's optimal trajectories of the 45 generation(N = 20, 60 % of good smell)

2.2) The ratio of good to bad is 45 % to 55 %.

In this case it is much more difficult for agent to reach the aim, because because he has to navigate in the area of a bad smell, and then to navigate in the area of a good smell.



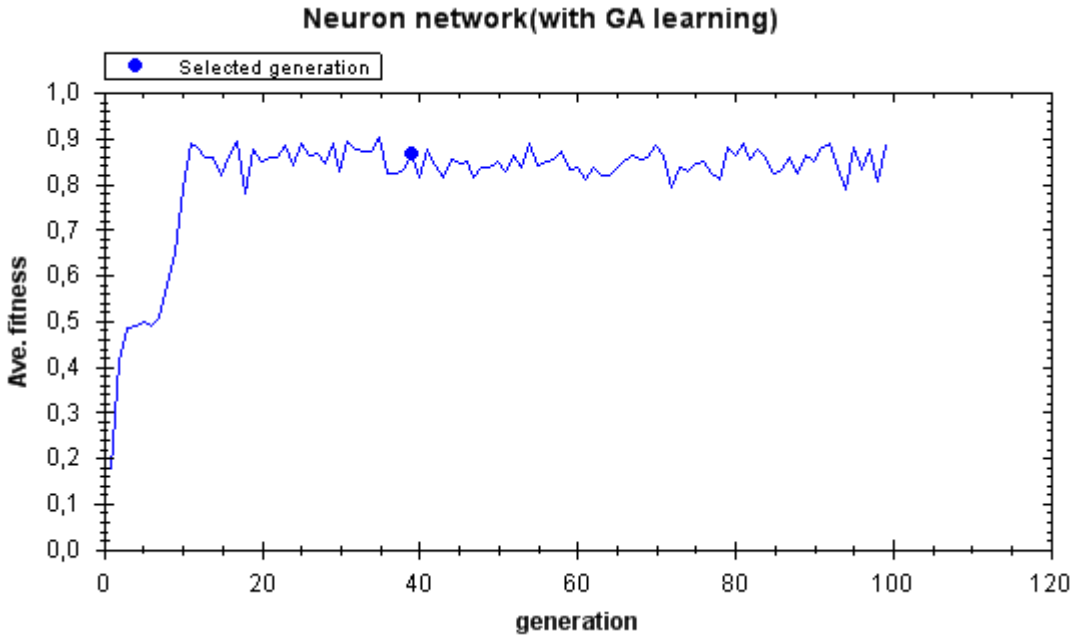
Picture 4.1.24. The dependence of the average fitness on the generation number(N = 20, 45 % of good smell)



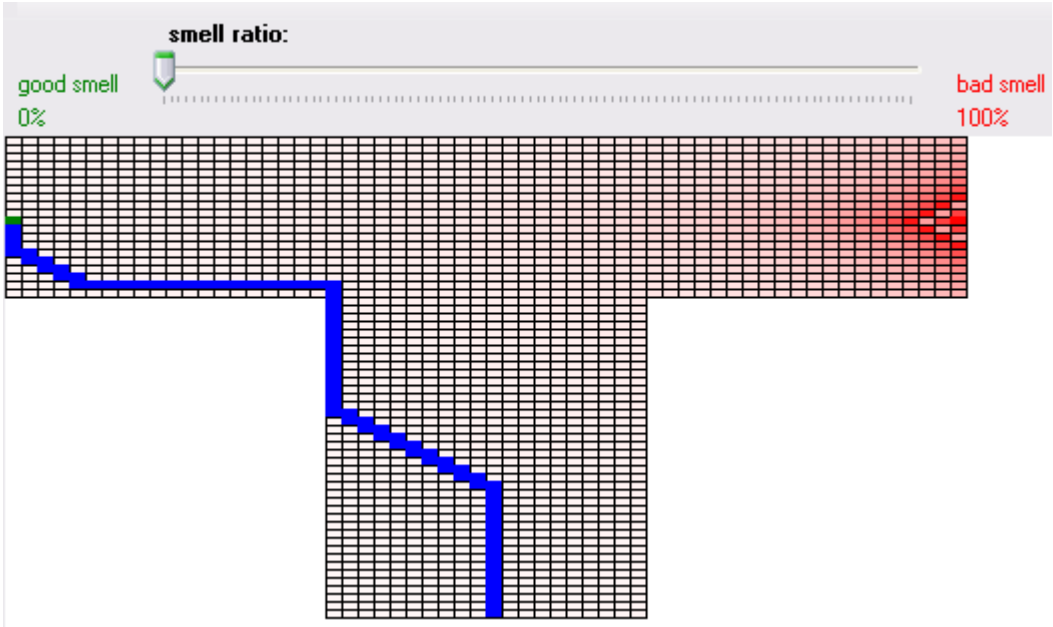
Picture 4.1.25. An example of the agent's optimal trajectory of the 100 generation(N = 20, 45 % of good smell)

2.3) The ratio of good to bad is 0 % to 100 %.

In this case agent has to find good aim by the bad smell. This task is much more easier than the previous one and is like the first test a bit, because good path is situated in the area of only one smell.



Picture 4.1.24. The dependence of the average fitness on the generation number(N = 20, 100 % of bad smell)



Picture 4.1.25. An example of the agent's optimal trajectory of the 40 generation(N = 20, 100 % of bad smell)

Results

- Multilayer neural network with genetic algorithm gives worse results than single-layer network. Such a network is more difficult to learn and the final results strongly depend on the initialization of the chromosomes.
- In case of success the algorithm always gives only the optimal results, the results do not depend on the muximum number of steps.
- The most difficult for the agent is a moment of transition between areas with different smells.
- The algorithm is able to provide for different sizes of the gridworld in average equally good results. At the same time, on the output we can take a family of different optimal paths.

4.2 Comparative analysis

The dependence of the optimal results on the size of the grid world:

- Random walk: yes
- Genetic algorithm: no
- Reinforcement learning: no(if agent uses the information about the distance information to the good aim)
- Neural network: no

Aspiration to extradition of optimum results:

- Random walk: no
- Genetic algorithm: yes(we must set the minimum required number of steps equal to 4 * N)
- Reinforcement learning: no (pretty close to optimal)
- Neural network: yes

Dependence on the maximum number of steps:

- Random walk: yes
- Genetic algorithm: yes
- Reinforcement learning: number of steps in the algorithm is unlimited
- Neural network: no

Suitability for using by real robots to solve the problem in real time:

- Random walk: no
- Genetic algorithm: no(maybe it is possible for constant external conditions; teaching in real time is pretty difficult)
- Reinforcement learning: yes
- Neural network: yes(but if the external conditions change strongly enough the agent have to continue teaching; teaching in real time is pretty difficult)

[1] J. H. Holland. Adaptation in natural and artificial systems. University of Michigan Press, Ann Arbor, 1975.
[2] Rosenblatt, F. (1962). *Principles of Neurodynamics*. Spartan Books.
[3] Sutton, Richard S. (1984). Temporal Credit Assignment in Reinforcement Learning