TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

IT40LT

Dmitri Kondratjev 164422IAPB

SOLVING THE ARTIFICIAL ANT PROBLEM USING GENETIC ALGORITHMS

Bachelor's Thesis

Supervisor: Margarita Spitšakova
PhD

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

IT40LT

Dmitri Kondratjev 164422IAPB

TEHISSIPELGA ÜLESANDE LAHENDAMINE GENEETILISTE ALGORITMIDE MEETODIL

Bakalaureusetöö

Juhendaja: Margarita Spitšakova
PhD

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Dmitri Kondratjev

21.06.2019

Abstract

Genetic algorithm is a heuristic search algorithm inspired by Darwin’s theory of natural evolution, which is commonly used to generate solutions for search and optimization problems.

The main purpose of this thesis is to develop a strategy for solving the artificial ant problem using Genetic Algorithms. To accomplish this task, the most common way of controller representation (finite state machines) was implemented and optimized, and several aspects of Genetic Algorithm were compared to find the optimum solution.

The result of this work is a search algorithm and an application that allows to control parameters of a search algorithm and visualize its results.

Python is used as a programming language. Genetic Algorithm is implemented without using external librariesю Graphical user interface is made with Pyglet.

This thesis is written in English and is 35 pages long, including 8 chapters, 24 figures and 1 table.

Annotatsioon

Geneetiline algoritm on otsingu algoritm, mis on inspireeritud Darwani evolutsiooniteooriast, mida kasutatakse optimeerimisprobleemidele ja otsimisprobleemidele lahenduste loomiseks.

Käesoleva töö põhieesmärgiks on arendada strateegiat tehissipelga ülesande lahendamiseks kasutades geneetisilisi algoritmisid. Töös vaadeldakse mitmeid geneetiliste algoritmide aspekte, et leida nendest kõige sobivamad. Töös kõige levinum viisi kontrolleri kujutamine on kasutatud ja optimeetitud (*finite state machines*).

Töö tulemuseks on otsingu algoritm ja tarkvararakendus, mis võimaldab geneetiliste parameetrite kontrollida ning otsingu algoritmi töö visualiseerida. Lisaks saab säilitatud tulemused reprodutseerida ning joonistada oma teid, kasutades neid otsingu algoritmi testimiseks. Kõik algoritmi tulemused on säilitatud *JSON*-formaadis.

Python on valitud programmeerimise keeleks. Geneetiline algoritm on implementeeritud ilma välise raamastikuta ja graafiline kasutajaliides on tehtud *Pyglet*'i abil.

Lõputöö on kirjutanud inglise keeles ning sisasldab teksti 35 leheküljel, 8 peatükki, 24 joonist, 1 tabel.

Table of Contents

List of abbreviations and terms

FSM	Finite State Machine
GA	Genetic Algorithm
GUI	Graphical User Interface
JSON	JavaScript Object Notation

List of Figures

List of Tables

1 Introduction

The theory of evolution by natural selection was formulated by Charles Darwin in his book “On the Origin of Species” in 1859. Evolution is the process in which living things change and develop over many generations. However, not all have equal chance for evolution. Darwin’s theory states, that better adapted individuals have more chances to survive [1].

In 1960, John Holland introduced Genetic Algorithms based on the concept of Darwin’s theory of evolution [2]. Relying on biological operators, Genetic Algorithms are commonly used for optimization and search problems.

This thesis deals with an optimization task for artificial ant problem.

1.1 Objectives

The main purpose of this thesis is to develop a strategy for artificial ant problem using Genetic Algorithms. It entails a choice of genetic parameters, finding of suitable fitness function and other optimization methods. Also, an implementation of application for the artificial ant problem’s simulation must be done.

1.2 State of the art

The artificial ant problem was first introduced by Jefferson et al. with usage of Genetic Algorithms. Their goal was to study several general aspects of Genetic Algorithms. They considered various parameters of Genetic Algorithm and suggested two approaches of genetic representation: finite state machines and artificial neural nets [3].

Later, the artificial ant problem was considered and popularized by Koza [4]. His approach was based on using of genetic programming. Since then many different methods were introduced and documented (Ant Colony Optimization [5], Evolutionary Strategies [6], Grammatical Evolution [7]).

2 Problem statement

Solving the artificial ant problem requires to find a strategy controlling an artificial ant so it could find all the food lying along an irregular trail. There are 3 common trails for this problem: “Santa Fe Trail”, “John Muir Trail” and “Los Altos Hills Trail”. The artificial ant along with the trail are located on the toroidal grid. Traditionally, starting position is the upper left cell of the grid with the artificial ant facing east [4].

The artificial ant has a very limited view of its environment along with the few numbers of actions. It has only one sensor that can detect one cell ahead and determine if there is food on it or not and it has no knowledge of its own position. The artificial ant can perform the following actions: turn right, turn left (while turning, the ant remains on the same cell), move forward or do nothing. If the cell, which the ant stepped on, has a piece of food, it will be eaten and removed from the grid [4].

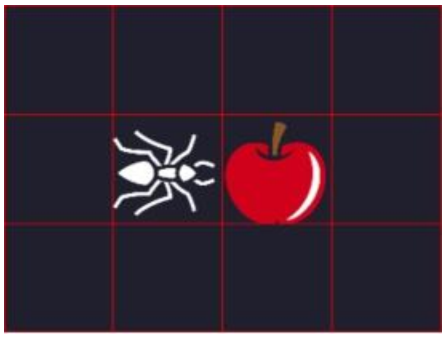


Figure 1. The ant can see food ahead

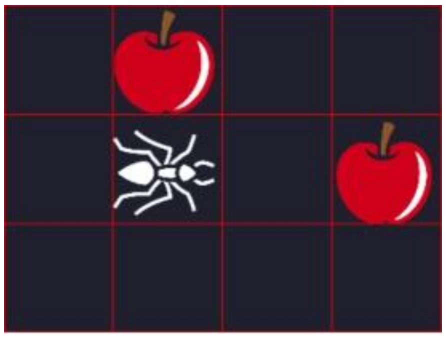


Figure 2. The ant cannot see food ahead

The artificial ant’s goal is to find and eat as much as possible pieces of food within a limited amount of time with the fact that each action requires 1 unit of time [4].

2.1 John Muir Trail

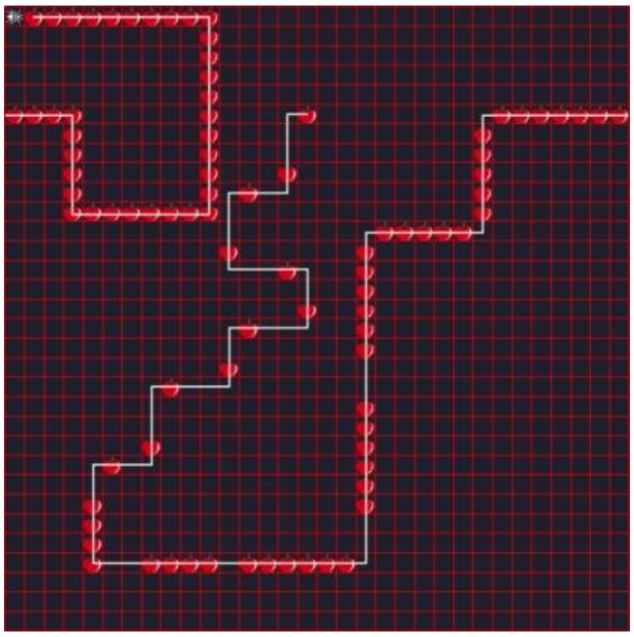


Figure 3. John Muir Trail

Figure 3 shows the “John Muir Trail” originally used for artificial ant problem. It contains 89 pieces of food, 20 turns, 4 single gaps, 7 double gaps and 7 triple gaps and a total length of 128. It requires a lower 148 action for the ant to eat all the food. It requires at least 148 action for the artificial ant to eat all the food. Traditionally, this trail is limited with 200 actions.

3 Genetic Algorithms

Genetic Algorithms (GA) are search methods inspired by evolution [8]. Each solution is encoded as a simple chromosome like data structure that can be modified with various genetic operators, imitating the process of natural evolution. GA's are often viewed as optimization algorithms; however, their actual scope of application is wider.

3.1 Canonical Genetic Algorithm

There are several types of GA's. To convey the main idea, canonical GA will be considered.

GA begins with initializing of initial population of individuals. Next step is to evaluate a fitness of every individual. GA is completed, if generation has an individual, that solves the problem according to its fitness, otherwise the new population is about to be generated. To generate the new population, some individuals are selected, using certain techniques of selection, to apply genetic operators to them (canonical GA considers crossover and mutation). After the new population is generated, second step repeats until the stopping criteria is satisfied [8].

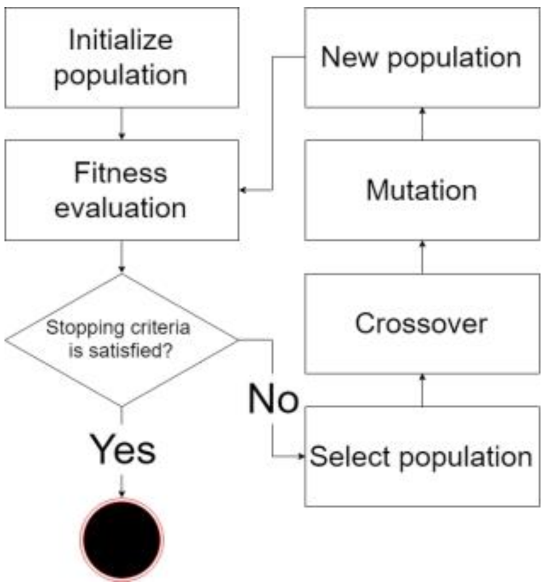


Figure 4. Canonical GA

3.2 Genetic representation

The most common genetic representation of GA is a sequence of bits, also called chromosome. Each bit of a sequence is gene, consequently chromosome consists of genes [8]. A distinctive feature of GA is that chromosome has a fixed length. It means, that applied genetic operators will not change a length of chromosome. This fact makes some genetic operators much easier e.g. crossover, that will be described below.

3.3 Genetic operators

The performance of GA depends on various operators.

3.3.1 Selection

The selection operator chooses from the current population individuals that will be allowed to reproduce the offspring for the next population. There are many different techniques of selection, but only truncation and tournament selections were used in this work.

The truncation selection is the simplest technique that orders individuals according to their fitness. Then, only certain number of fittest individuals (e.g. 10% - 50%) are selected for breeding. This selection method ensures that the best genetic material of the current population will be used in the next population. However, there is also a chance of rapid convergence [9].

The tournament selection is a variation of rank-based selection. Random individuals are selected from population and in accordance with their fitness, the fittest one is chosen for breeding. The number of selected individuals for tournament is determined by the size of the tournament. This selection method adds more diversity of genetic material, because there is a chance that even individual with the fitness lower than average can be chosen for breeding [9].

Nevertheless, there is chance, that selected for breeding individuals will not produce better individuals and consequently, the best genetic material will be lost. To avoid such scenario, elitism is used. Elitism means that the best individual in the population is maintained in the next population, so it will not be lost while selection and following genetic operations.

3.3.2 Crossover

The crossover operator plays a vital role in GA. The main idea of this operator is to combine two chromosomes to produce the offspring. Single-point crossover is one of the simple techniques. It splits chromosomes at the randomly selected crossover point and swaps right sides of split chromosomes between themselves as shown below [10].
Before single-point crossover:

- Chromosome 1: 1001|101010
- Chromosome 2: 0101|110011

After single-point crossover:

- Offspring 1: 1001|110011
- Offspring 2: 0101|101010

Two-point crossover operates as single-point crossover, but it picks 2 points and swaps genes between selected points. Uniform crossover produces offspring from uniformly selected genes of 2 chromosomes. Each gene has equal probability of being exchanged [10]. Figure 5 shows the influence of crossover on fitness. It is seen, that uniform crossover is more chaotic in the beginning, however, since the 25th generation there are no big differences. Two-point crossover will be chosen for tests since it showed stable growth.

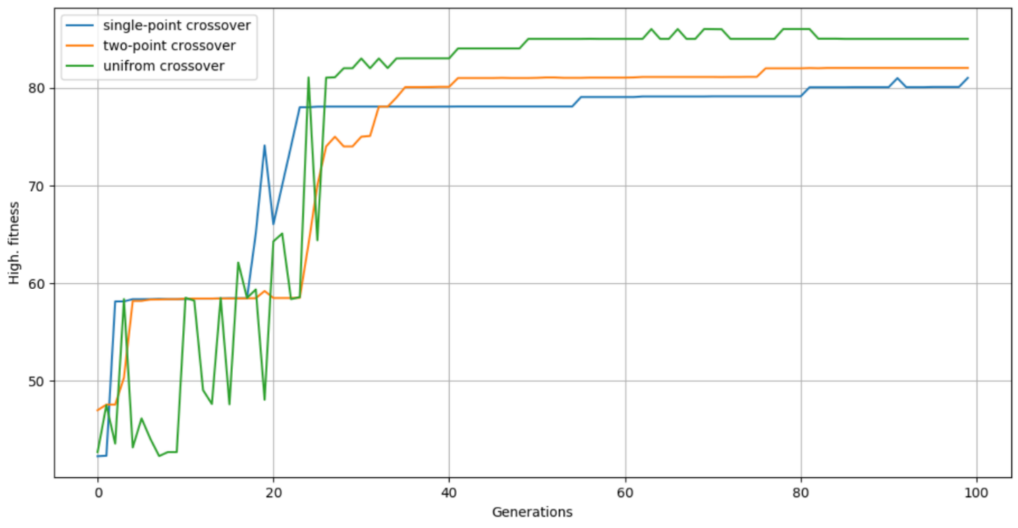


Figure 5. Comparing of crossover methods

3.3.3 Mutation

The role of the mutation operator is to change the genes of the offspring to increase diversity of the population. The mutation rate controls the frequency of genes to be mutated and affects the efficacy of GA. Figure 6 shows the work of GA with mutation rate set to 10%. It is seen that with high mutation rate, GA turns into random search algorithm. In this case finding of solution is based on a pure luck. In figure 7 mutation rate is set 0.5%, which is low enough to not destroy the best solution.

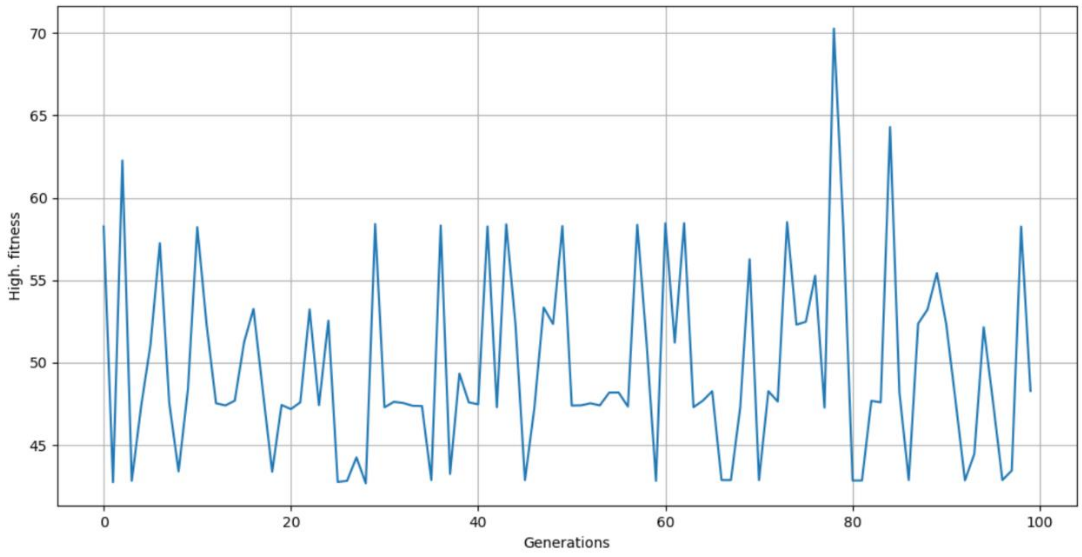


Figure 6. GA with the mutation rate of 10

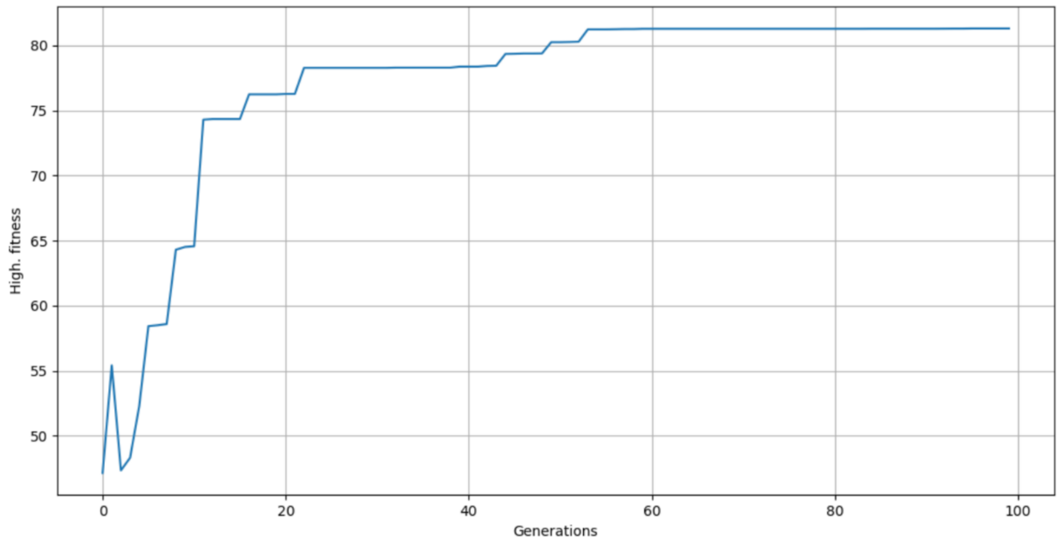


Figure 7. GA with the mutation rate of 0.5

4 Finite state machines

Finite state machines (FSM) are one of the most powerful models, which are used to describe behavior [11].

4.1 Moore and Mealy machines

Two best known models of FSM are Moore and Mealy machines. There is no a single opinion, which machine is better, and choice depends on certain needs [11].

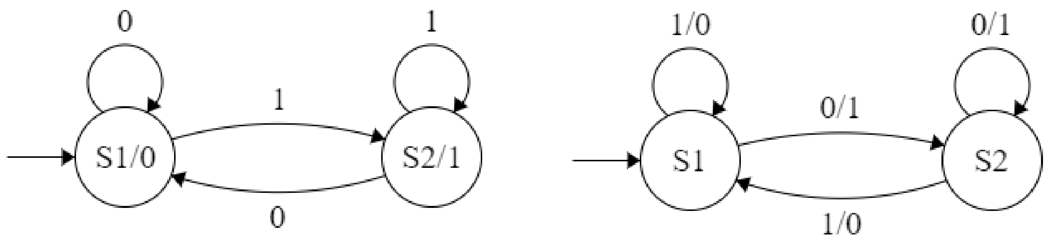


Figure 8. Moore and Mealy machines

Figure 8 shows 2 FSM’s, Moore machine on the left and Mealy machine on the right. Moore machine has only entry action, while Mealy machine has only output action. Mealy machine is the most common way to describe behavior of the artificial ant, that was used in many researches related to GA [3] [4].

5 Methodology

5.1 Tools

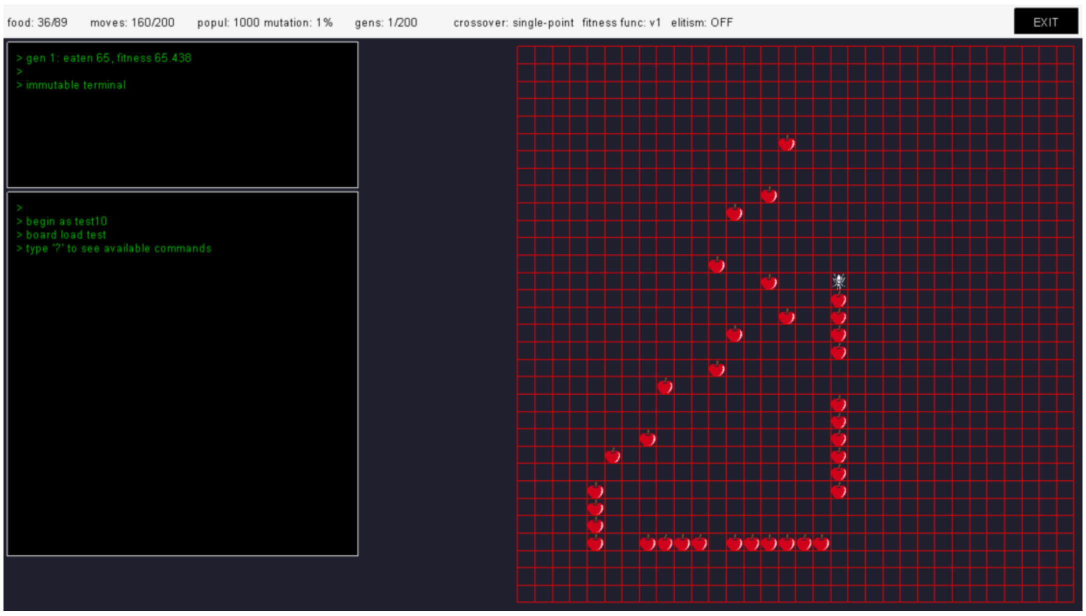
Python was chosen as a programming language for implementation of all aspects of this thesis. This choice is based on the author's experience. According to the GitHub search system, there are much more repositories, that uses Python for GA [12]. However, no external libraries were used for implementation of GA, this part of coding was done with pure Python.

Python also was used to build up a graphical user interface (GUI). For this purpose, Pyglet was used. Pyglet is a cross-platform windowing and multimedia library, intended to develop visually rich applications [13].

There are several reasons why implementation of GA was done with pure Python. One of the reasons is that in such way it is more comfortable to connect between each other implementation of GA and GUI. Also, it is easier to create a convenient structure of GA, while learning of external library may be a longer process and may have such obstacles as a limitation of functionality.

5.2 Application

Figure 9 shows a screenshot of developed application within this work, which is used as test environment. Functionality of this application allows to edit trails, set genetic parameters, visualize run of fittest individual in the population, replay saved data. Control system is implemented as terminal. For example, the command "begin as test" starts GA and when it is finished, the result is saved to "test.json" file. All the possible commands can be seen by pressing "?" mark.



```
food: 36/89     moves: 160/200     popul: 1000 mutation: 1%     gens: 1/200     crossover: single-point  fitness func: v1  elitism: OFF
```

```
> gen 1: eaten 65, fitness 65.438
>
> immutable terminal
```

```
>
> begin as test10
> board load test
> type '?' to see available commands
```

Figure 9. Application for the artificial ant problem

6 Implementation of Genetic Algorithm

6.1 Chromosome representation

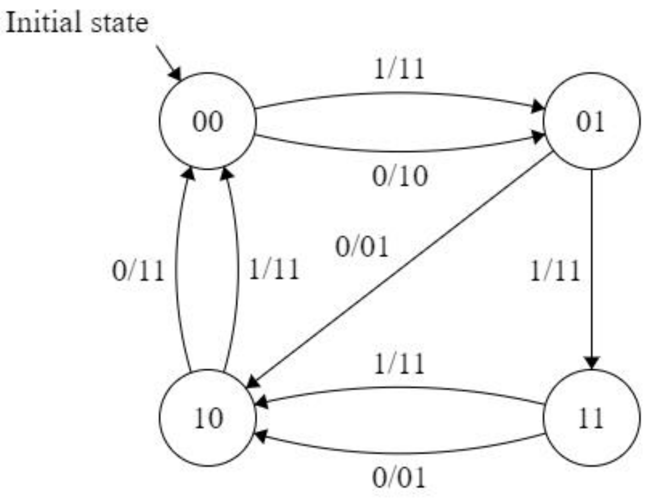


Figure 10. FSM with 4 state, which describe a behavior of the artificial ant

To use finite state machine as chromosome, it must be encoded. Figure 10 shows the FSM with 4 states that describes an artificial ant's behavior. Each state may have 1 or more inputs and exactly 2 outputs. Output is used to describe artificial ant's sensor, "1" if sensor detects food ahead and "0" if not. According to the output, FSM returns a corresponding action and proceeds to the next state. Actions are defined as strings of bits: "01" - turn right, "10" - turn left, "11" - move forward and "00" - do nothing. For all FSMs, that were generated in this work, initial state is always 0.

Table 1. State transition table

Old state	Input	Output action	New state
00	0	10	01
00	1	11	01
01	0	01	10
01	1	11	11
10	0	11	00
10	1	11	00

11	0	01	10
11	1	11	10

Figure 1 shows the state transition table which describes the FSM given above. For chromosome representation only 2 last columns are required, because the first column is FSM's sorted serial number and the second column values repeat for each state. Thereby, first state can be encoded with the following way: 1001 1101, where first 4 bits are for transition if output's value is 0 and last 4 bits for output's value 1. First 2 bits of each transition describes action and last 2 bits is the serial number of next state. Figure 11 shows the result of encoded FSM.

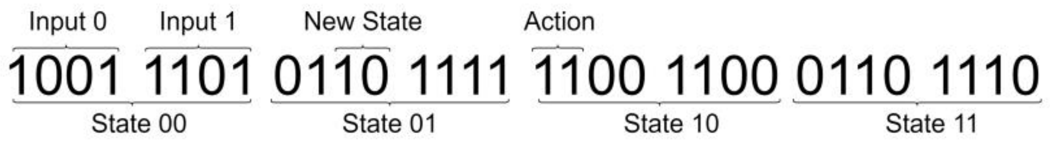


Figure 11. Encoded finite state machine

6.2 Chromosome length

The chromosome length depends on the amount of states of FSM and can be calculated by the following formula: $2 \times (2 + n) \times 2^n, n \in N$. n is number of bits required to encode one state of FSM, therefore if a state is encoded with 2 bits, then FSM will have $2^2 = 4$ states and chromosome length will be $2 \times (2 + 2) \times 2^2 = 32$ bits.

However, having 2 states is not enough to solve the problem, because it has low search (space 2^{32} possible solutions). In related research FSM with 32 states (each state encoded with 5 bits) was used [3]. It seems to be an optimal option and will be used in this thesis. Encoded chromosome length is 448 bits and the total number of possible generated FSM's is 2^{448}. It is worth mentioning, that having 32 states does not mean that all of them will be used. There are also can be unreachable or unvisited states by an artificial ant during its simulation.

6.3 Initial population

As a rule, initial population is randomly generated, however heuristic initialization can be used. The generation of initial population is implemented with the following principle: while transitions are generated randomly bit by bit, output action are randomly chosen from the following set: {01, 10, 11}. In this way, combinations of 00 will not be generated. As 00 is defined as “do nothing”, it has negative effect on individuals’ behavior and consumes 1 unit time.

For the artificial ant problem, we can define in advance that to move forward, if the ant is seeing a piece of food, seems to be an effective tactic. Therefore, it was decided to generate 20% of all chromosomes with “if food ahead – move” genes. This method will increase an average fitness of initial population, however there is highly low chance, that final solution will be found within first generation.

6.4 Isomorphic machines

Among the population can be easily determined chromosomes with the same gene set. However, there are also can be found individuals, who have same behavior, but different chromosome representation. These chromosomes may be a result of two isomorphic FSM’s.

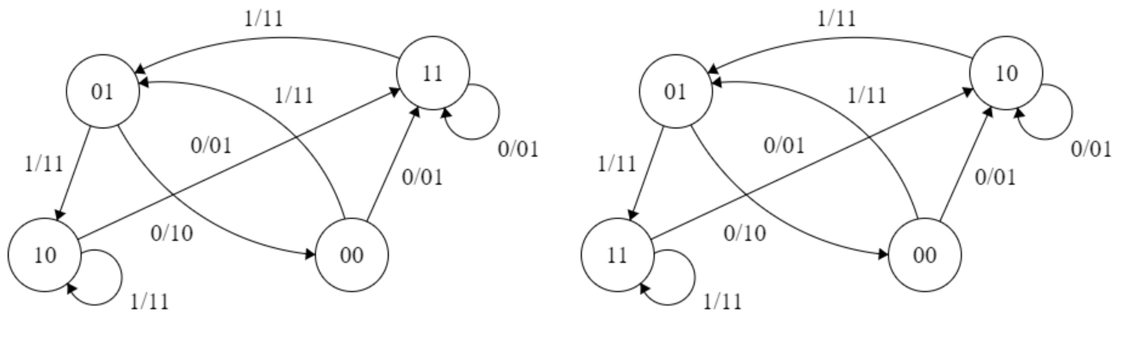


Figure 12. Example of 2 isomorphic FSM’s

Figure 12 shows an example of two isomorphic FSM’s. Their gene set is different, because two states (10 and 11) are swapped, but transitions are equivalent.

In figures 13 and 14 are shown the influence of isomorphic FSM’s. Having high number of isomorphic FSM’s, search space was reduced, so it was hard to produce the offspring, that would have better gene set. However, the situation has changed on generation 57. The author assumes, that a sequence of advantageous mutations had happened, so the number of isomorphic FSM's dropped and, as result, evolution continued.

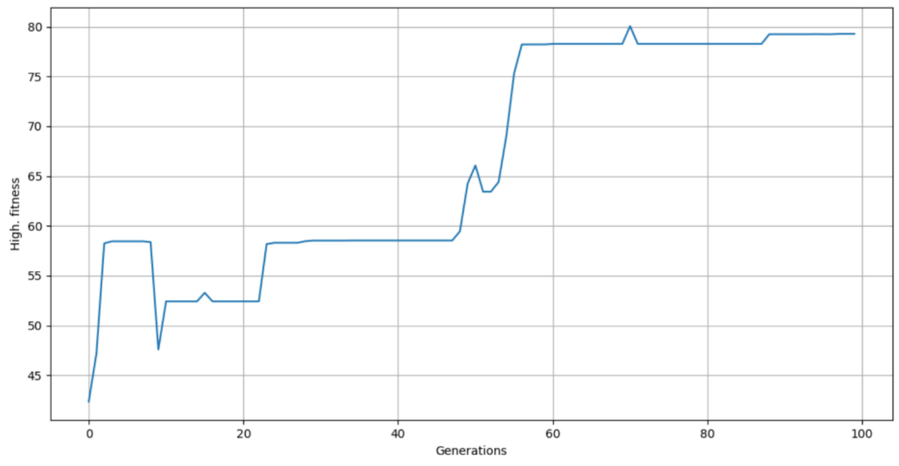


Figure 13. Influence of isomorphic finite state machines on fitness

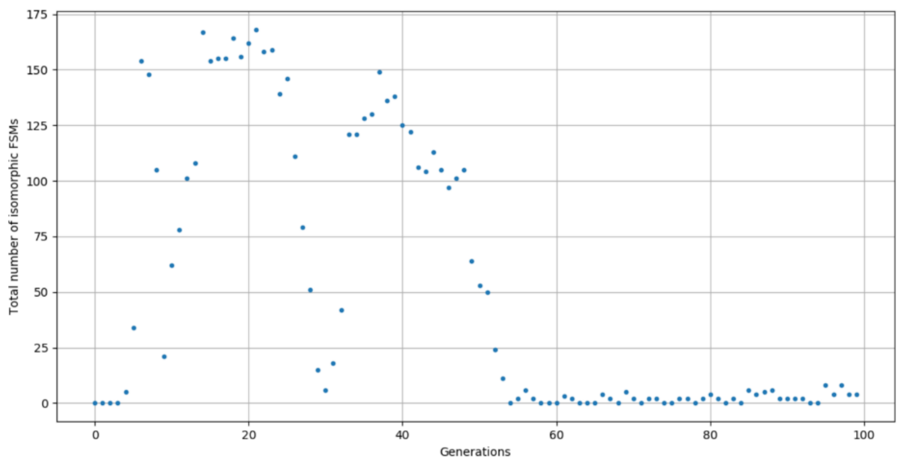


Figure 14. Total number of isomorphic state machines among generations

In this section will be proposed that way to minimize appearing of isomorphic FSM's. In figure 15 is shown the algorithm that was developed to determine problematic FSM's. Below follows a brief description of this algorithm.

Starting at initial state, algorithm visits each next state and adds output actions with index of current state to the list. Initial index is 0, at the end of algorithm, each state has unique index. Visiting of states happens recursively. If current state is already visited, output actions and index of current state add to list once again (it is important to check transition where old state is new state as well), then current recursion is ended. For all isomorphic FSM's this list of actions is the same, so they now can be easily determined.

```
def sequence_of_actions(chromosome):
      actions = []
      visited = {}
      states = chromosome.get_states()
      def get_recursive(curr_state, index):
            if curr_state in visited:
                  index = visited[curr_state]
            else:
                  while index in visited.values():
                        index += 1
            actions.extend([states[curr_state][NO_FOOD][ACTION], \
                           states[curr_state][FOOD][ACTION], index])
            if curr_state in visited:
                  return
            visited[curr_action] = index
            get_recursive(states[curr_state][NO_FOOD][NEXT_STATE], index)
            get_recursive(states[curr_state][FOOD][NEXT_STATE], index)
      get_recursive('0' * bits_in_state, 0)
      return tuple(actions)
```

Figure 15. Algorithm for sequence of action determining

6.5 Fitness functions

Fitness function defines how good an individual from the current population solves the problem. Various fitness functions can influence a GA in different ways [14].

A fitness function for the artificial ant problem depends on success of trail passing. As the main purpose is to eat all the food on the grid, it could be suggested to use amount of eaten food as fitness value. However, there are could be many ants, who have the same fitness value, but a run was different. For example, two ants ate 10 pieces of food, that were in a straight line. The first ant moves forward if sees food, so he did it within 10 moves. The second ant turns 360 degrees every time when he sees food and then moves forward, so it takes 46 moves to eat all the food. Obviously, that the second ant has the worse chromosome, so fitness functions should be more complex. The aim of this section is to produce several fitness functions for further study. Each function contains a value of eaten food and additional aspect to create competition.

Fitness function #1 uses a value of used states of FSM.

$$fitness = all\ eaten\ food + \frac{total\ states - used\ states}{total\ states}$$

Fitness function #2 uses an amount of performed turns left or right.

$$fitness = all\ eaten\ food + \frac{total\ moves - \ turns}{total\ moves}$$

Fitness function #3 is defined by a step of last eaten piece of food.

$$fitness = all\ eaten\ food + \ \frac{total\ moves - last\ eaten\ peice\ of\ food}{total\ moves}$$

Combinations of fitness functions are also considered and can be obtained by the following method:

$$fitness = \frac{fitness_1 + fitness_2 + \cdots + fitness_n}{n}$$

7 Experiments

7.1 Genetic parameters

According to the previous description and comparing of genetic operators, was made a choice to use the following configuration:

- Selection: Truncation
- Generation limit: 200
- Population size: 500
- Crossover: Two-point
- Mutation rate: 0.5
- Elitism: True
- Trail: John Muir Trail

Generation limit decision was based on results of related research [3]. In this research best FSM with 13 states was generated in generation 200.

7.2 Statistics collection

After every generation, required statistics is collected to be visualized and processed. Statistic contains highest fitness value among population, average fitness value of population, amount of used states by individual with highest fitness value, total number of isomorphic FSM's. All the data is storing in JSON format. Evolution process of the best individual from every generation can be replayed using developed for this thesis application.

7.3 Fitness function analysis

For each fitness function 10 independent runs were performed.

7.3.1 Fitness function #1

The main idea of the first fitness function is to reduce number of used states of FSM. 1 of 10 individuals (3) was able to eat all the food within 190 generations. Others have completed their runs with results between 81 and 88. Average value of used states is 12,6. Average score is 84,8.

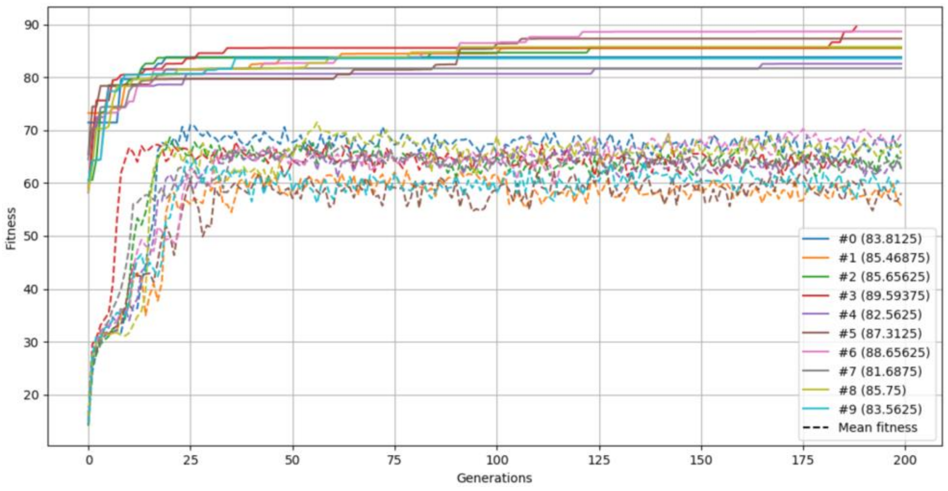


Figure 16. Results for fitness function #1

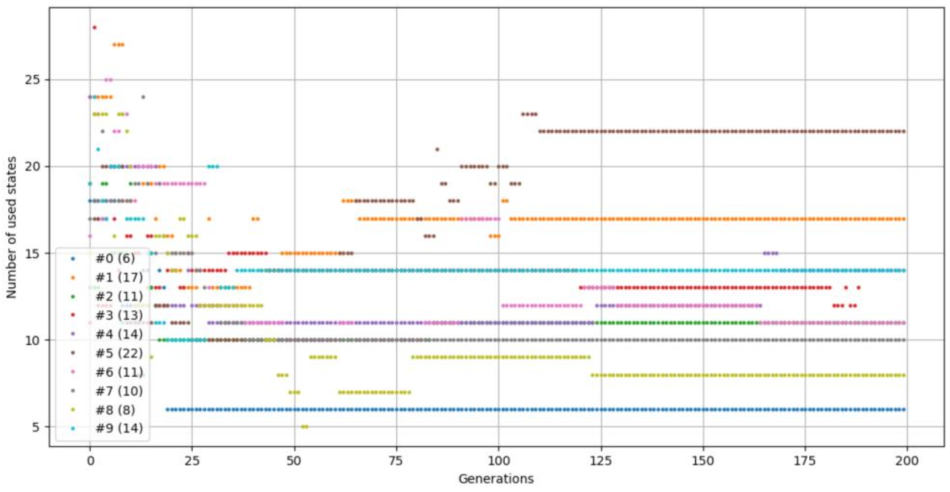


Figure 17. Number of used states for fitness function #1

7.3.2 Fitness function #2

The second fitness function should encourage individuals with a smaller number of rotations. 3 individuals (2, 5, 4) have completed their run with the maximum amount of eaten food. Average value of used states is 19,3.

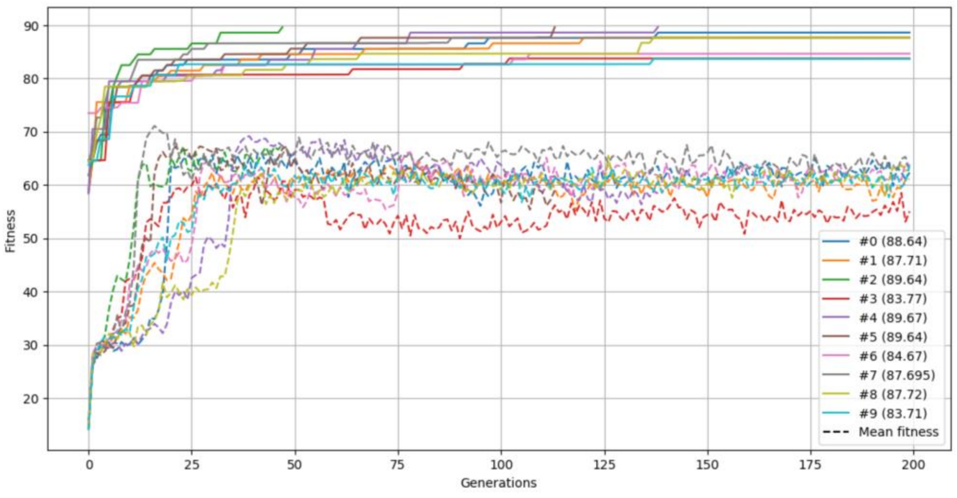


Figure 18. Results for fitness function #2

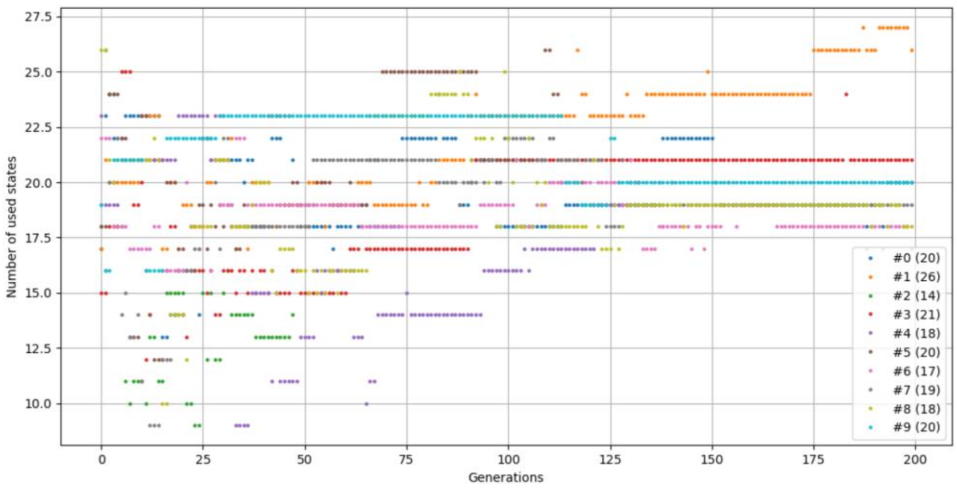


Figure 19. Number of used states for fitness function #2

7.3.3 Fitness function #3

The third fitness function is the most efficient. This function encourages individuals, who have eaten more food with a smaller number of actions. 7 of 10 runs are successful. Fastest result (9) was gotten within 45 generations by FSM with 18 states. Another 5 results (1, 5, 8, 7, 5) were gotten before 100th generation and another 1 (8) before 115th generation. The smallest amount of used states is 10. Average value of used states is 19,4.

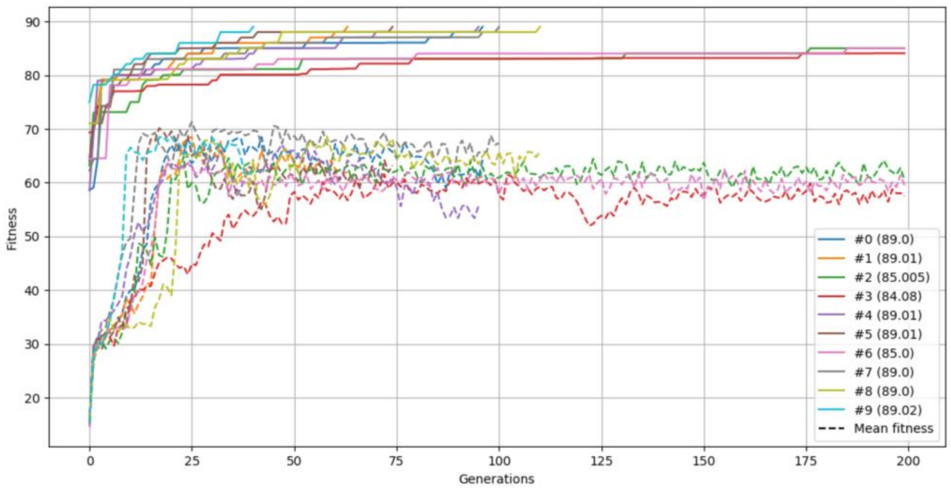


Figure 20. Results for fitness function #3

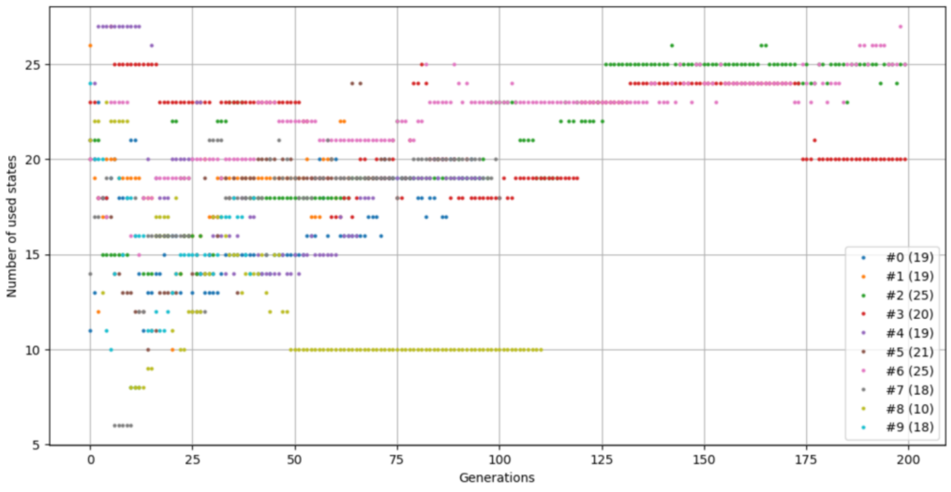


Figure 21. Number of used states for fitness function #3

7.3.4 Fitness function #1 and #3

Fitness function with the best results was combined with the first fitness function in order to reduce an amount of used states. Average value of used states is 11,9 and average score is 85. The only one successful run ended with 14 used states.

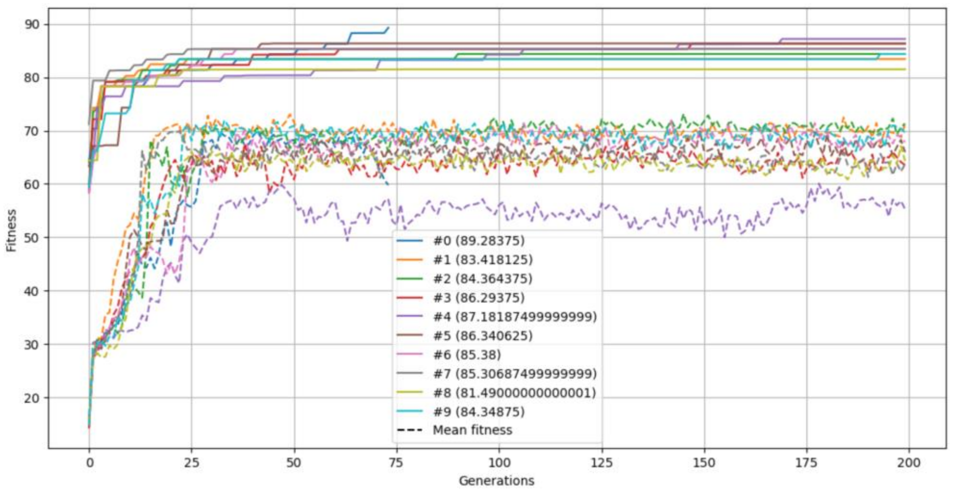


Figure 22. Results for fitness functions #1 and #3

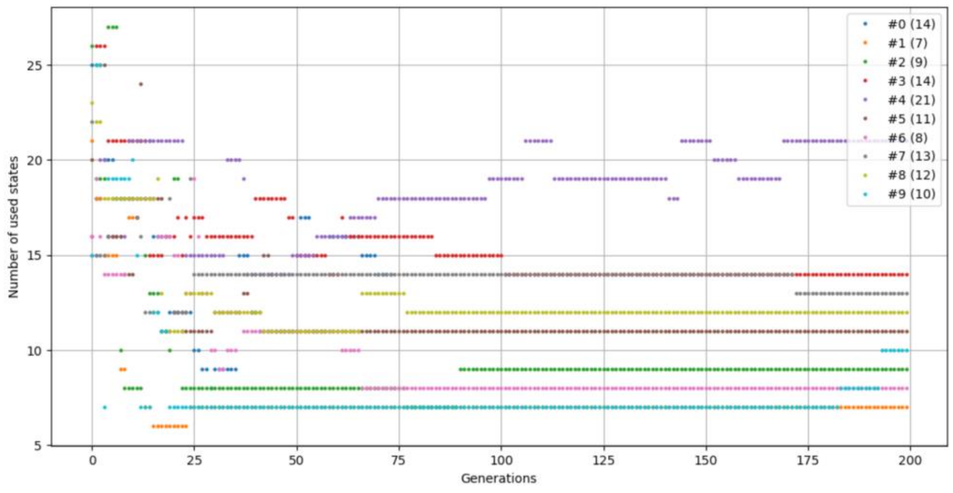


Figure 23. Number of used states for fitness function #1 and #3

7.4 Conclusion

Proposed fitness function for reduction of used states (#1) has a bad performance for artificial ants regarding another considered functions. It is hard to receive better results with a lower number of states, which would successfully complete the trail. Combination of fitness functions #1 and #3 gives slightly better results.

The reason why fitness function #3 showed better results that fitness function #2 is that fitness function #3 covers several aspects of successful run. Completing the trail with lower amount of action means, that must be performed less turns, less gaps between eating.

Jefferson et al. presented FSM with 13 states, which takes 200 steps to successfully complete the John Muir Trail [3]. This FSM was obtained in generation 200. To compare, all the successful FSM’s in this work were received before generation 200. Also, FSM with 10 states was obtained using fitness function #3. It takes 199 steps to complete that trail.

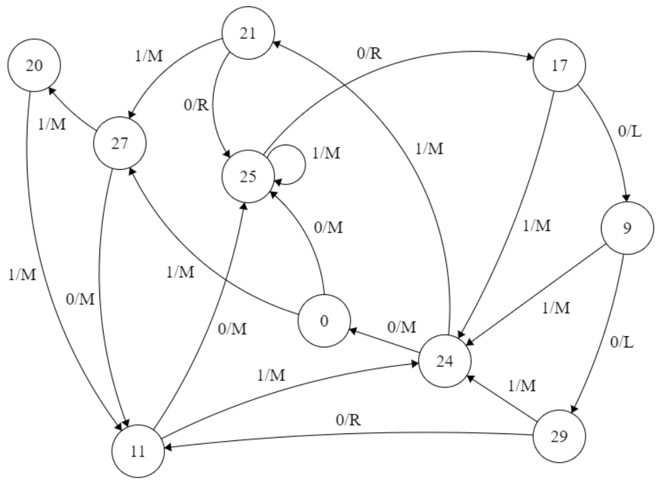


Figure 24. FSM with 10 states, which scores 89

8 Summary

The main goal of this this thesis was to develop a strategy for solving the artificial ant problem using Genetic Algorithms. For this, an existing way for controller representation was considered and optimized for better performance.

Proposed strategy successfully managed to solve the problem by completing the task with maximum score 7 of 10 times. Also, was found a better finite state machine than one which was introduced in related research [3].

Another goal was to develop an application for the considered problem, that was also accomplished.

For future work, different genetic representations can be considered. Developed application can be further improved to be able to start several GA at once.

In conclusion, the main objectives of this thesis were successfully achieved.

References

[1] C. Darwin, On the Origin of Species, John Murray, 1859.

[2] "Genetic algorithm," Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Genetic_algorithm. [Accessed 16 May 2019].

[3] Jefferson, D., Collins, R., Cooper, C., Dyer, M., Flowers, M., Korf, R., Taylor, C., and Wang, A., Evolution as a theme in artificial life: The Genesys/Tracker system, 1991.

[4] J. Koza, Genetic programming - on the programming of computers by means of natural selection, MIT Press, 1992.

[5] Dorigo, Marco & Birattari, Mauro & Stützle, Thomas, "Ant Colony Optimization: Artificial Ants as a Computational Intelligence Technique," *IEEE Computational Intelligence Magazine,* vol. 1, pp. 28-39, 2006.

[6] Spears, W.M. and Gordon, D.F, "Evolving finitestate machine strategies for protecting resources," *Proceedings of the 12th International Symposium on Foundations of Intelligent Systems,* p. 166–175, 2000.

[7] Karim, M.R. and Ryan, C., "Sensitive ants are sensible ants," *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference,* p. 775–782, 2012.

[8] M. Mitchell, An Introduction to Genetic Algorithms, MIT Press, 1998 .

[9] Khalid Jebari, Mohammed Madiafi, Abdelaziz Elmoujahid, "Parent Selection Operators for Genetic Algorithms," *nternational Journal of Engineering Research & Technology,* vol. 2, no. 11, pp. 1141-1145, 2013.

[10] A.J. Umbarkar, P.D. Sheth, "CROSSOVER OPERATORS IN GENETIC ALGORITHMS: A REVIEW," *ICTACT JOURNAL ON SOFT COMPUTING,* vol. 6, no. 1, pp. 1083-1090, 2015.

[11] Wagner, F., Schmuki, R., Wagner, T., Wolstenholme, P., Modeling Software with Finite, 2006.

[12] GitHub, [Online]. Available: https://github.com/search?q=genetic+algorithm. [Accessed 12 May 2019].

[13] "pyglet Documentation," pyglet, [Online]. Available: https://pyglet.readthedocs.io/en/pyglet-1.3-maintenance/index.html. [Accessed 12 May 2019].

[14] M. Mitchell, An Introduction to Genetic Algorithms, 1998.

[15] Daniil S., Chivilikhin, Vladimir I. Ulyantsev, Anatoly A. Shalyto, "Solving Five Instances of the Artificial Ant Problem with Ant Colony Optimization," *7th IFAC Conference on Manufacturing Modelling, Management, and Control,* vol. 46, no. 9, pp. 1043-148, 2013.