Gliwice 21.06.2021r.

Biologically inspired artificial intelligence

Project Report

“Prisoner’s Dilemma”

Authors of the project

Paweł Mika gr. GKiO1

Ryszard Kałuziński gr. GKiO1

1. Introduction

The aim of the project was to create and implement an evolutionary algorithm that learns the strategy of conduct in the „Prisoner's dilemma" which is a problem in game theory. It is based on a two-player, non-zero sum game in which each player can gain by betraying (defecting) their opponent, but both will lose if both of them betray (defect). Prisoners can also cooperate with each other and in case they both cooperate - both will gain more than if they both betray, but if the second prisoner betrays when trying to cooperate - you can lose a lot more. An example of the prisoner's dilemma payoff matrix is presented below.

Prisoner B / Prisoner A	**Prisoner B stays silent (*cooperates*)**	**Prisoner B betrays (*defects*)**
Prisoner A stays silent (*cooperates*)	Each serves 1 year	Prisoner A: 3 years Prisoner B: goes free
Prisoner A betrays (*defects*)	Prisoner A: goes free Prisoner B: 3 years	Each serves 2 years

Source: https://en.wikipedia.org/wiki/Prisoner%27s_dilemma

2. Problem analysis

The outcome of a single "dilemma" can be completely random, so the real problem is the iterated prisoner's dilemma, which is playing the dilemma multiple times.

There are many different strategies related to the game that have been published by people involved in game theory. One of the most effective strategies is "Always Defect", which means that regardless of the opponent's moves, the player will always choose to betray. Another popular strategy is tit for tat, where the player cooperates on the first move, and on each subsequent move he does what his opponent did in the previous move.

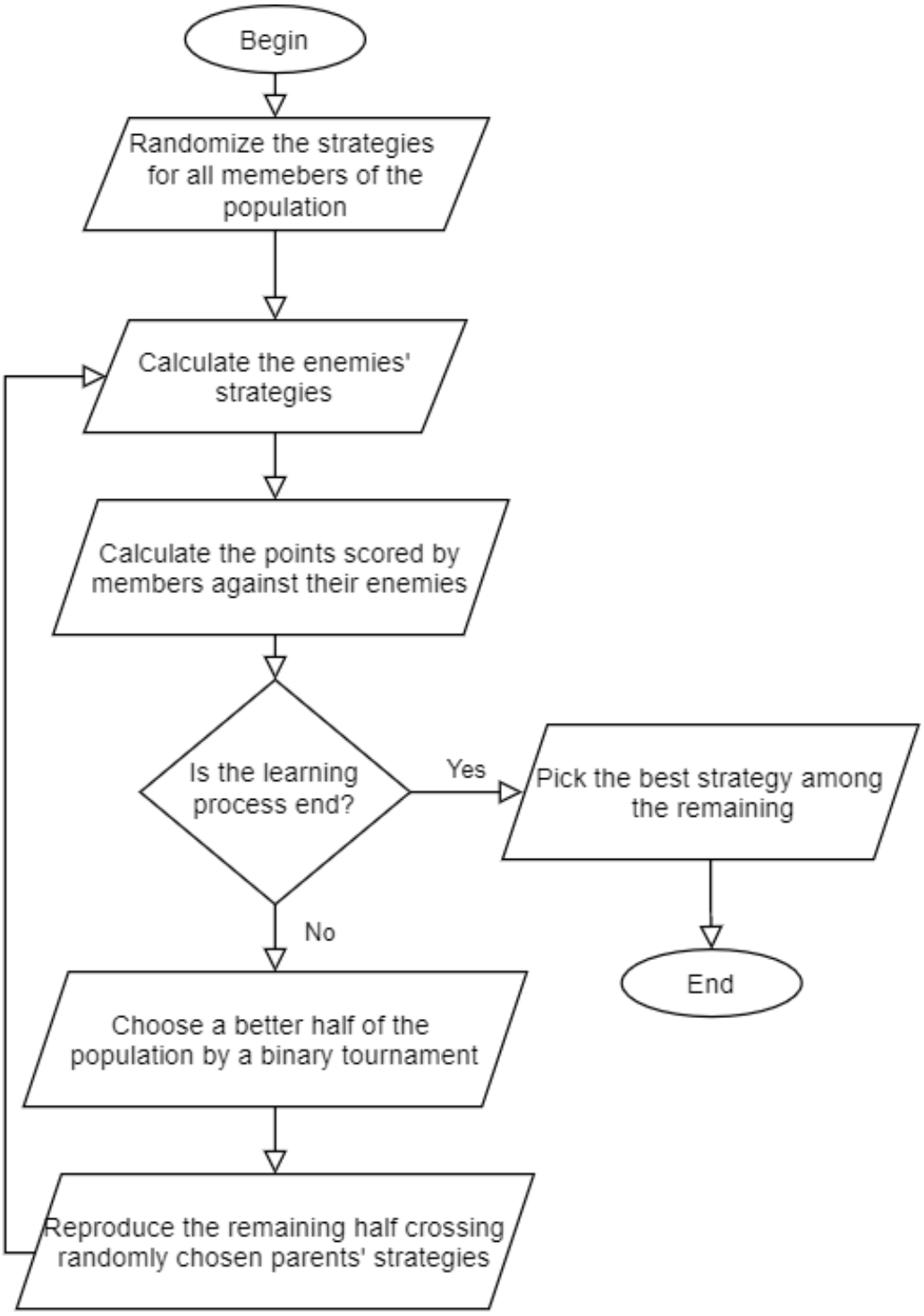
The aim of the project is to create an algorithm that, by playing games against established strategies, will learn how to score as many points as possible against a specific strategy while minimizing the points scored by the opponent (the largest possible point difference between the player and the opponent).

The first idea to solve the task seems to be checking all possible strategies one by one, depending on the number of turns in the game.

All of the strategies and their effectiveness against the opponent's strategy should be compared and the best strategy at the moment should be stored in cache. This solution may make sense in the case of a short game (several / a dozen turns), but with longer games, the time to find the best strategy increases exponentially. In the case of a game consisting of 10 moves, 2^{10} different strategies should be analyzed, but in case of a game in which there are, for example, 50 turns, there are 2^{50} different strategies. Adding one move doubles the amount of computation needed for this approach.

A much better idea seems to be to use NSGA-II sorting (Non-dominated sorting genetic algorithm II), which not only allows you to significantly accelerate finding the best tactics, but also allows you to focus on 2 attributes at the same time (the largest possible point difference and the largest possible player's score). In this approach, a strategy is drawn at the very beginning for each member of the test population. Then a binary tournament is organized in which the strategies of randomly selected members of the test population are compared with each other. The winner of the fight is placed in the group of parents of the next generation. The tournament goes on until the amount of winners is equal to half of the population. After the tournament is over, the remaining half of the population is made up by mixing the tactics of randomly selected parents. After creating the missing half of the population - each of the new members has one move randomly changed so as not to limit the possibility of finding the best strategy to the combinations of the strategies drawn at the beginning. After the population is replenished to its original size, the tournament is played again and the cycle repeats itself for a predetermined number of generations during which the algorithm has to learn the best strategy. After all generations are over, the best strategy is chosen from those that remain.

The flowchart of the algorithm for finding the best strategy is presented below.



3. Internal and external specification

The algorithm was implemented using the C ++ objective language. The commented code is attached to the report. The program implements the above-mentioned NSGA-II algorithm, as well as functions needed for the correct operation of the algorithm (e.g. a function to draw the initial strategy and a function to calculate the opponent's strategy depending

on the player's strategy). The project was implemented in form of a console application in which the user is asked to enter the following parameters:

• The size of the testing population

• The number of turns in one game

• The number of generations (how much time does the program have to find the best strategy)

• Opponent's tactics

After entering the above parameters, the program starts the learning process and every 100 completed generations it informs the user for how many generations the learning process has been going on.

After the learning process is done, the program lists all of the moves of the best strategy it could find. At the end, the program counts the points scored by its strategy, the points scored by the opponent, and then calculates their difference and states who won and with what advantage (or a draw if there was one).

4. Testing

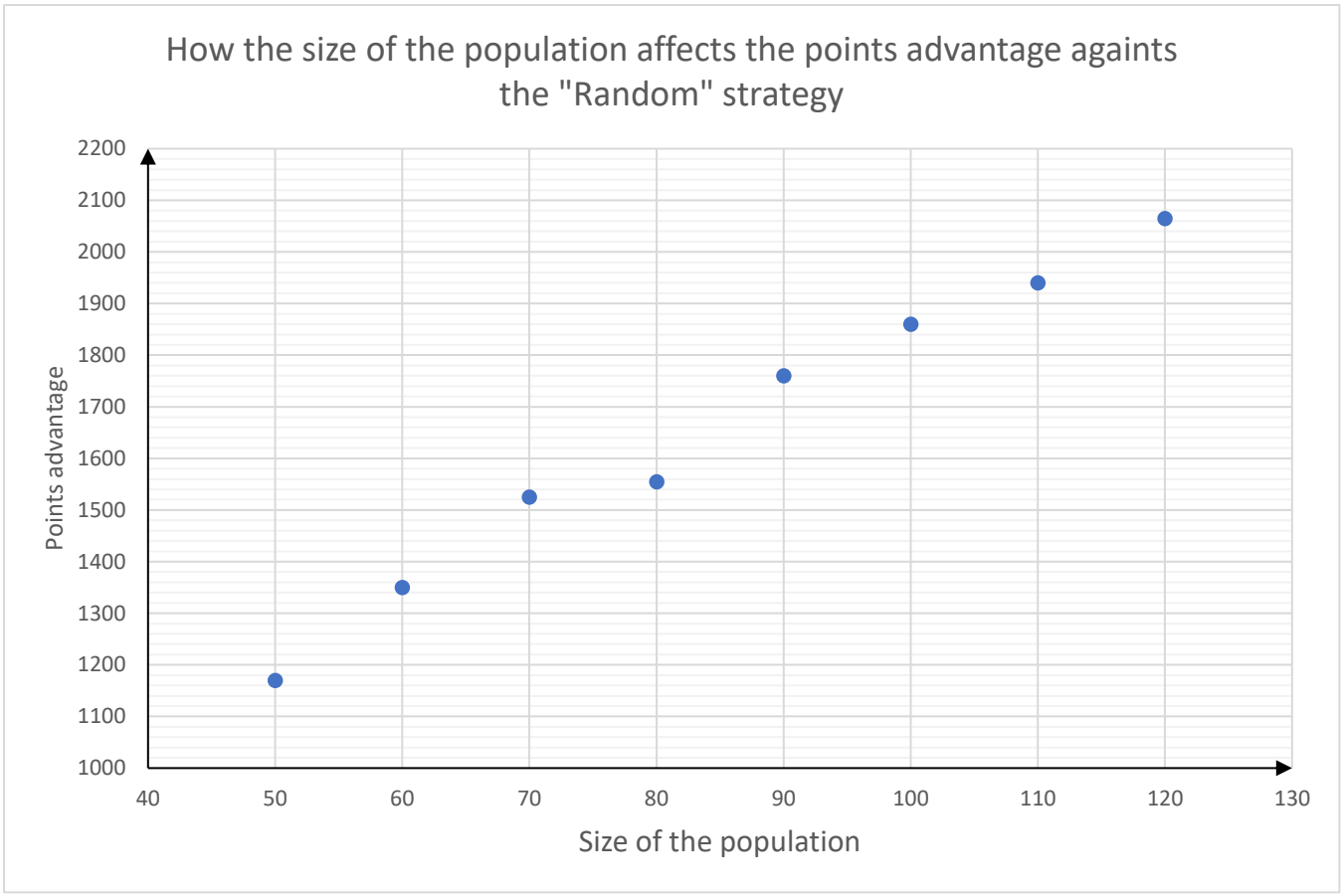
In order to test the correctness of the algorithm's operation and the effectiveness of its learning, the parameters listed in the third point were changed. The opponent's tactics that have been implemented are:

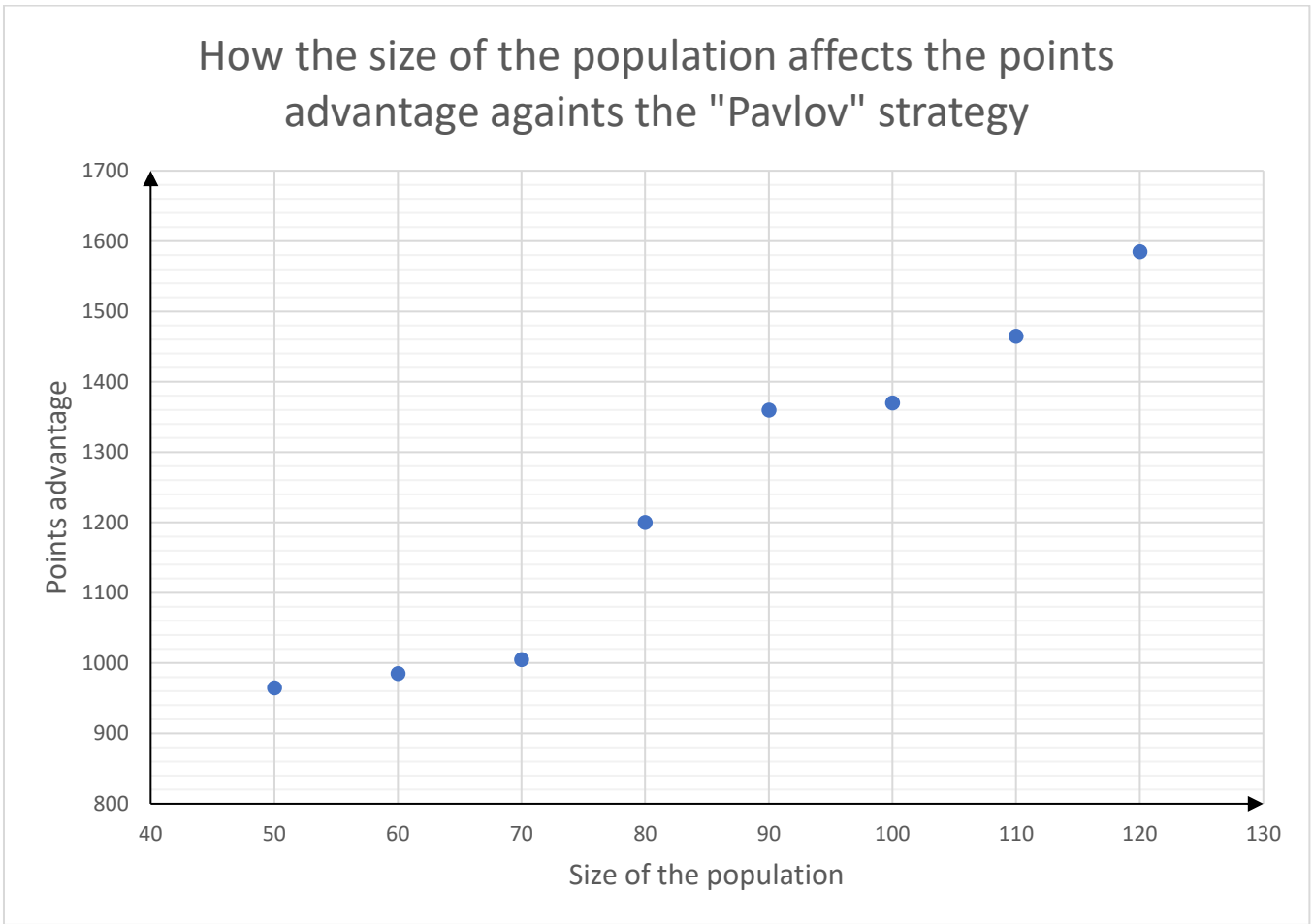
- Random - The tactic is that the strategy adopted by the opponent is randomly selected using pseudo-random numbers
- Always Cooperate - The tactic is that the opponent always cooperates
- Always Defect - The tactic is that the opponent always defects
- Tit for Tat - The tactic is that the opponent starts with cooperation, and in each subsequent move he does what his opponent (player) did in the previous one
- Suspicious Tit for Tat - The tactic is that the opponent starts with betrayal, and in each subsequent move he does what his opponent (player) did in the previous
- Imperfect Tit for Tat - The tactic is that the opponent starts with cooperation, and in each subsequent move with a certain

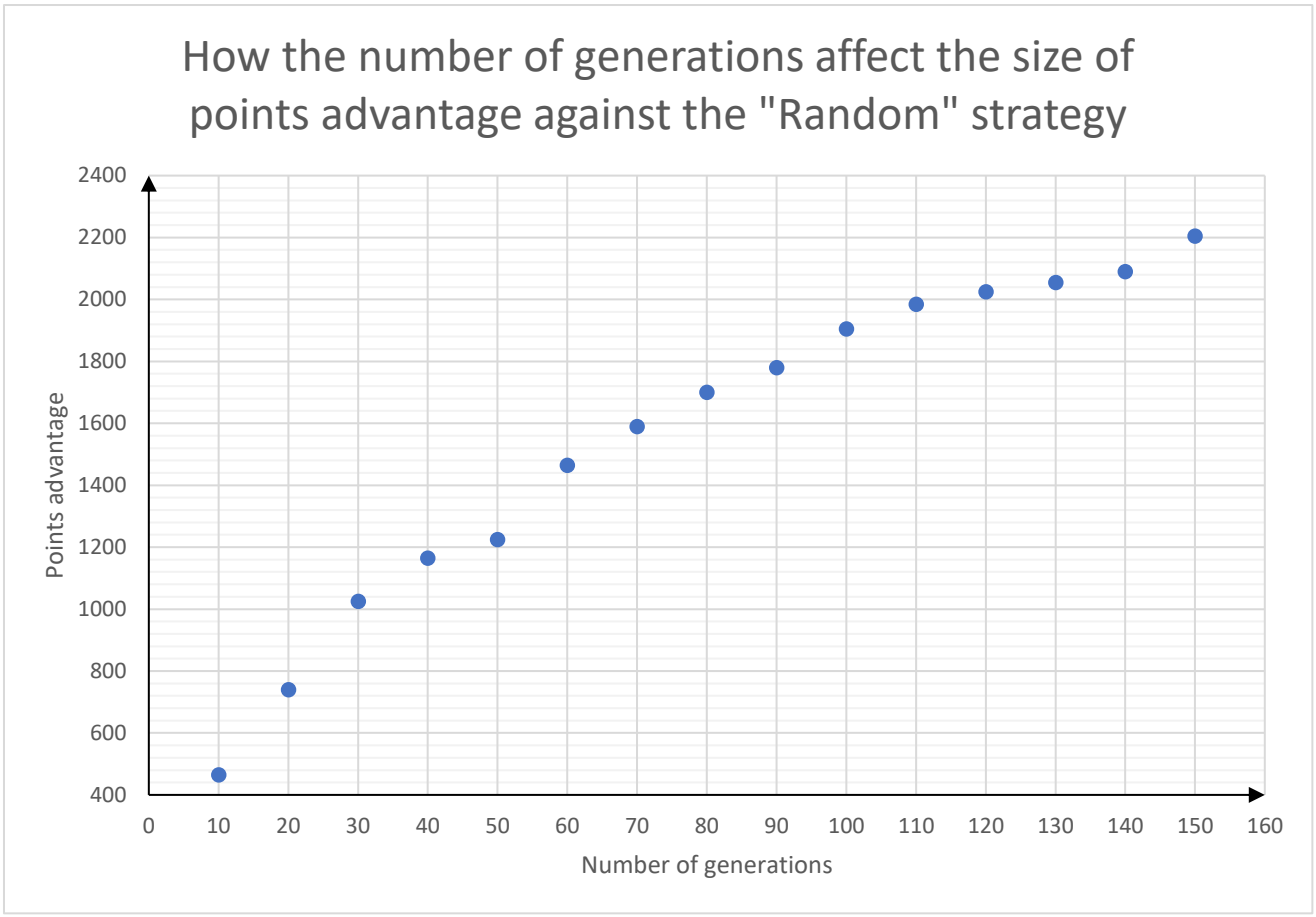
probability, he does what his opponent (player) did in the previous one

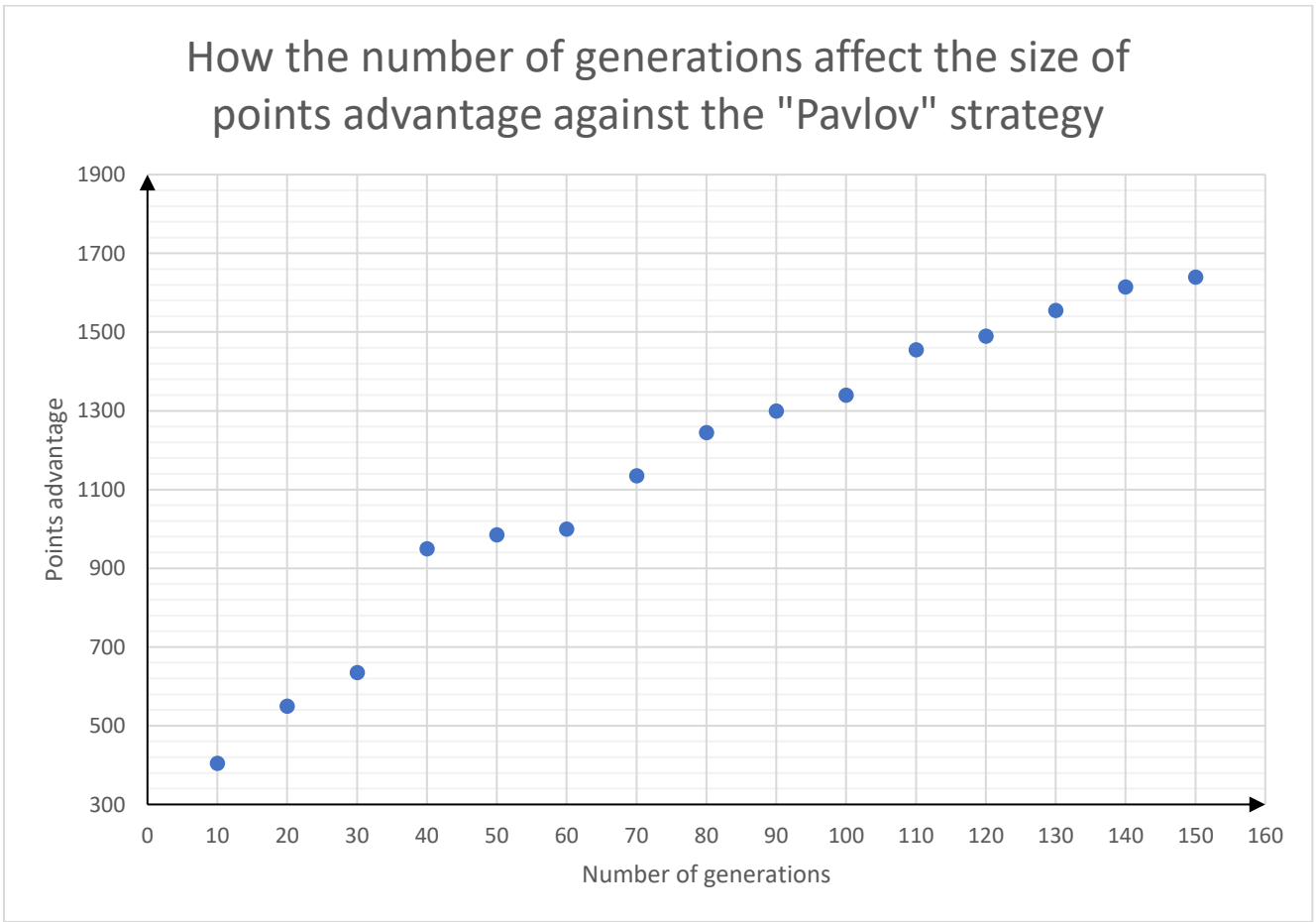
- Pavlov - The tactic is that the opponent starts with cooperation, and in each subsequent move:
    - Cooperates if both players did the same move in the previous turn
    - Defects if players did something different in the previous turn

By comparing our genetic algorithm with different tactics and changing the numbers of population sizes, the number of game rounds and the number of generations during which the algorithm searches for the best strategy, we noticed that in any case where the best strategy exists - the algorithm with the right amount of time to learn is able to find it. When the opponent's moves are somehow random (e.g. imperfect tit for tat), the best tactic does not exist, therefore the bigger number of generations and the greater size of the population do not always mean the bigger size of the points advantage over the opponent. Different results can also be obtained by changing the value of the prizes for individual situations, because apart from the maximum difference in points, the algorithm also aims to achieve the highest possible own result. The graphs showing the relationship between point advantage on the opponent and the size of the population or the number of generations with other values constant are presented below:


How the size of the population affects the points advantage againts the "Random" strategy
Points advantage
2200
2100
2000
1900
1800
1700
1600
1500
1400
1300
1200
1100
1000
40
50
60
70
80
90
100
110
120
130
Size of the population


How the size of the population affects the points advantage againts the "Pavlov" strategy
Points advantage
1700
1600
1500
1400
1300
1200
1100
1000
900
800
40
50
60
70
80
90
100
110
120
130
Size of the population


How the number of generations affect the size of points advantage against the "Random" strategy
Points advantage
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
Number of generations


How the number of generations affect the size of points advantage against the "Pavlov" strategy
Points advantage
1900
1700
1500
1300
1100
900
700
500
300
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
150
160
Number of generations

„Random” and „Pavlov” strategies are the opponents, againts which the best strategy is that the player should always defects, for against „Random” it is the most secure option that ensures the highest gains, and against „Pavlov” it is just the best strategy considering the values set by us for every outcome.

5. Summary, conclusions

The project was a great success. We've managed to create an algorithm that, in a reasonable amount of time, is able to devise the best strategy for the "Prisoner's Dilemma" game (provided there is one). The time it takes to develop the best strategy depends on the number of moves in the game and your opponent's strategy. For more complex opponent strategies, that involve the element of randomness, increasing the time spent on learning does not necessarily mean that the performance increase as a lot depends on how your opponent plays against the best strategy found. The most important element of the course of the project was to choose the right algorithm, because the effectiveness of the finished program greatly depends on it. The choice of the environment should be consistent with the preferences of the people writing the program, so that when implementing the algorithm it is possible to focus on the essence of its operation, and not on looking for functions and libraries in the documentation of a given programming environment / language. Completing the project gave us a sense of relief, and knowing that the program is working properly is utterly satisfying.

6. Sources

- https://www.sciencedirect.com/topics/computer-science/crowding-distance
- https://www.sciencedirect.com/science/article/pii/S1877705811022466
- https://www.youtube.com/watch?v=SL-u_7hIqjA
- https://towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3
- https://www.iitk.ac.in/kangal/seminar/shashi.pdf

- https://www.lesswrong.com/posts/BY8kvyuLzMZJkwTHL/prisoner-s-dilemma-with-visible-source-code-tournament
- https://www.sciencedirect.com/science/article/pii/S2405896316309661
- https://plato.stanford.edu/entries/prisoner-dilemma/strategy-table.html

7. Link to the files

https://drive.google.com/drive/folders/1oY8Jdw7Ko2hhpltlMlb7yTTjU7qQrrBO?usp=sharing