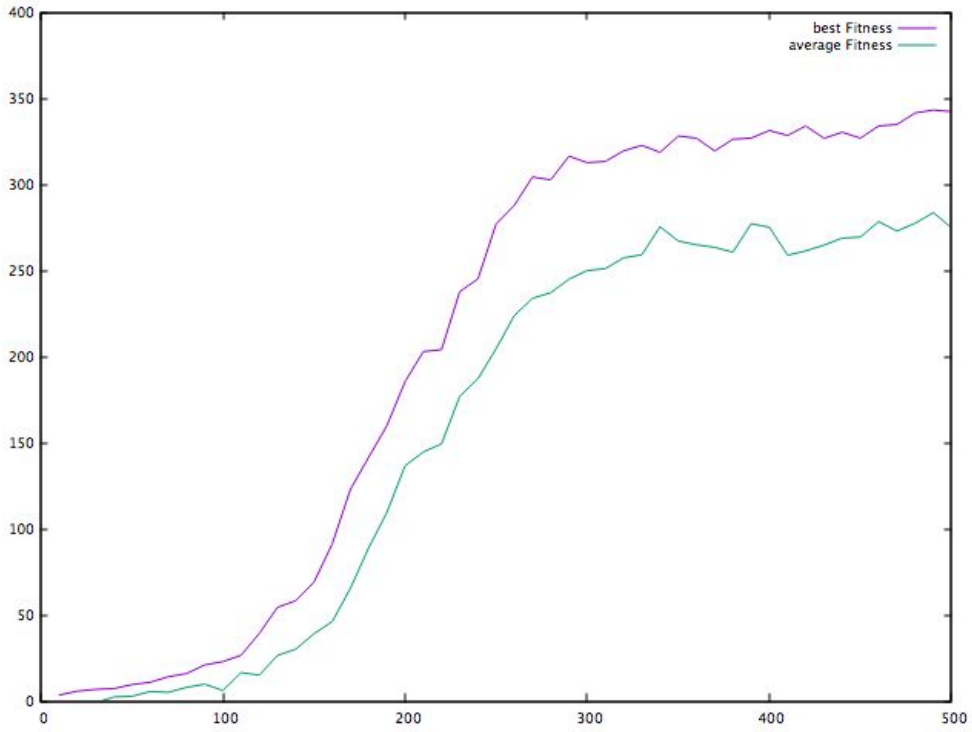
Bio-Inspired Artificial Intelligence
Report - Assignment 2 - 02/24/17
Robby the Robot
Dr. Prof. J. Marshall
Alexander Jermann

Addendum - 03/01/27

During conference, I was advised to change the Rank Selection function (see code assign02.py), since it was doing complicated and inaccurate calculations, which were probably responsible for the my low best fitness of 387.71 after whole 1680 generations. The new version of the code uses the Weighted Choice function used in the simple genetic algorithm, to select genomes using weights. The Rank Selection function uses a list filled with all numbers from 1-200 in consecutive increasing order once, as weights and calls weighted choice using sorted genomes and weights. The result of the adapted code was a higher slope (faster increase or less computations to find a good strategy). Sadly no better fitness was found, but it must be noted that the new code works a lot more efficiently.



Introduction

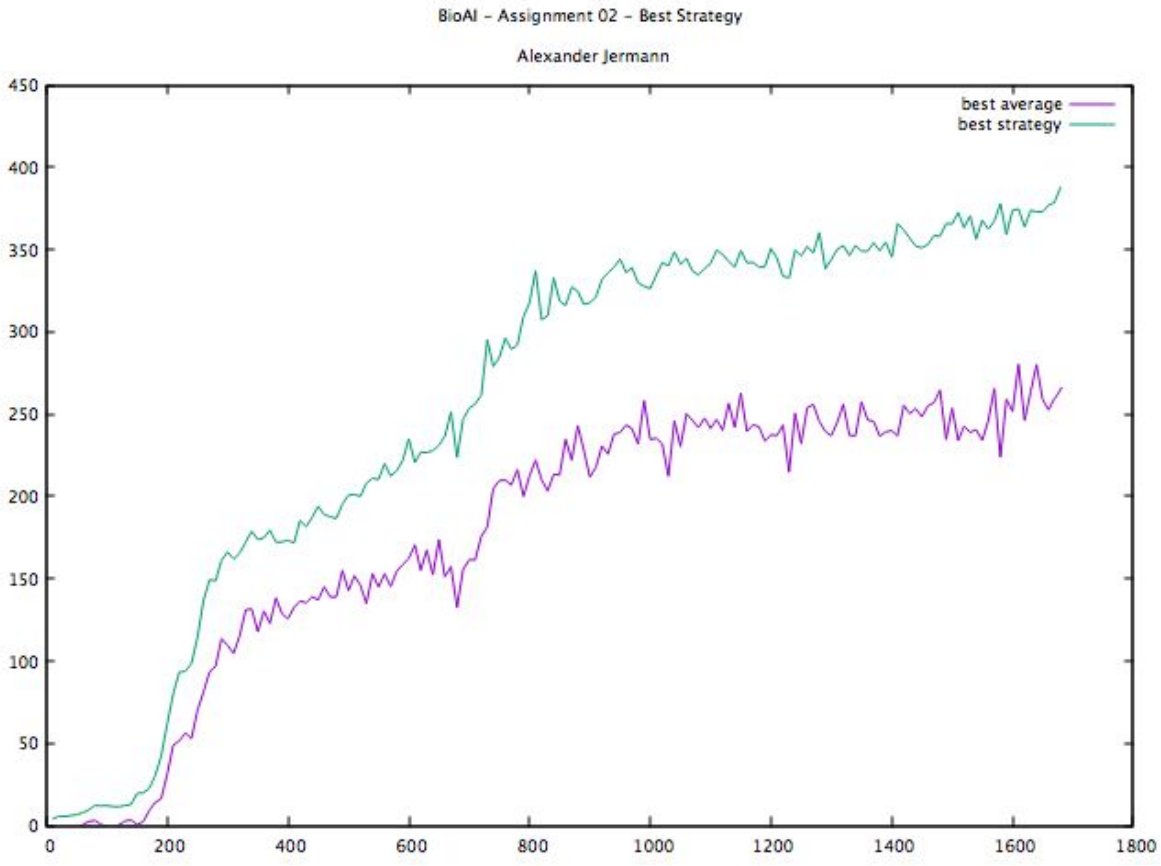
The goal of this assignment is to implement a genetic algorithm to evolve a strategy for a robot to pick up randomly distributed cans in a 10 x 10 grid. At any given position the robot can only differentiate between three different objects (Wall, Can, Empty) laying in five different directions (Up, Down, Right, Left and Center) which robby encodes into a percept string ‘wwce’. There is a maximum of 243 situations the robot can find himself in (some are not possible in a 10 x 10 grid, like ‘wwwww’). We set the genome to be 243 characters long where each locus represents an action for the robot to perform in any given situation. The genetic algorithm generates a population of randomly generated genomes or control strategies, performs 100 cleaning sessions for each genome in the population and takes an average to measure the fitness of each genome to create a separate fitness list. A new population is then built by crossover over and mutation using rank selection.

Reflection

Having implemented a simple genetic algorithm for last assignment made this assignment a lot more approachable. The similarity in the structure made understanding the concept easier. One function I struggled with in this assignment was the implementation of the Rank Selection function. In my first experiments I was getting ever decreasing negative fitness values instead of the other way around. The way I solved it was by reading the ‘sorted fitness values’ list from the back (highest value) to the front (lowest value). The rank selection seems to be working now based on the results.

Executing the program resulted to be a lot harder than expected. In order to obtain a high fitness value I executed the GA 20 times. Each generation took an average of 15 seconds to compute, and thus it took in average a little over 2 hours to compute 500 generations. In total my GA ran for 47 hours.

Best Strategy



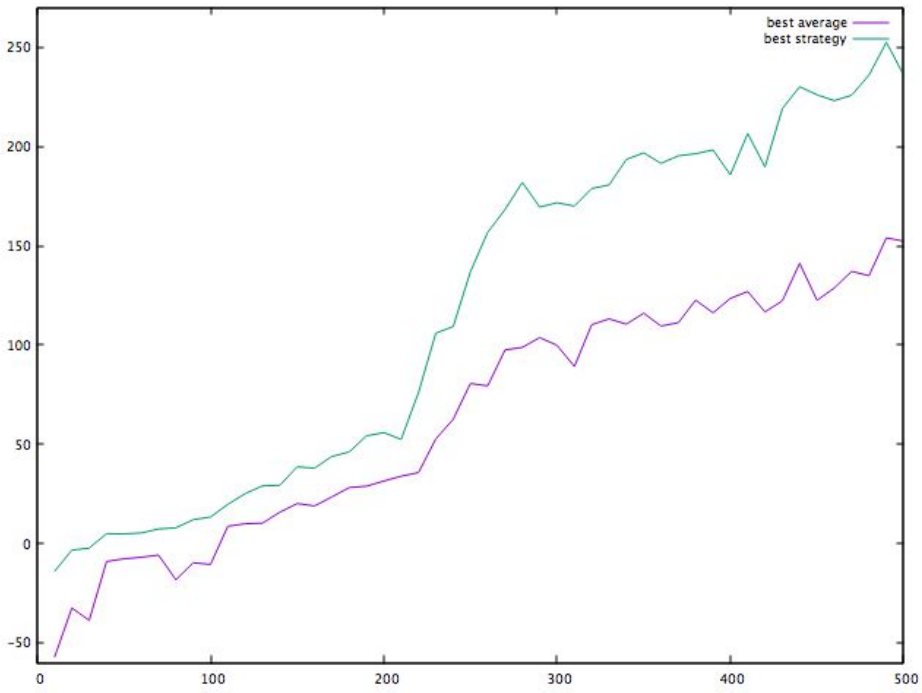
As depicted on the graph it took more than 500 generations to find a strategy with a high average fitness value. After 500 generations the best strategy never had a fitness value over 300. In my most successful run the best strategy for robby was found at generation 1680 with a fitness value of 387.71. The best strategy for robby that I was able to compute is:

```
0563361522553312563516526623521161513525332516623324452511352526512152536355451663533552612523351363541156413566424533515242261565413646534152646512463566645454253513341532563311523523662443151351526131341514634123362555114325632612333342226 31
```

The best strategy was found using the following parameters:

```
Population size: 200
Crossover rate: 0.8
Mutation rate: 0.005
Actions per cleaning session: 200
Number of generations: 500
```

GA Output



The first GA program I ran, I used the following parameters:

```
Population size: 200
Crossover rate: 1.0
Mutation rate: 0.005
Actions per cleaning session: 200
Number of generations: 500
```

The first generations had a negative fitness value, but after 200 generations the fitness started increasing at a fast rate. The best fitness value in this first run was 236.00. A problem that I often bumped into was the fact that after a certain amount of generations, the average fitness of the population would not increase and just oscillate in a range of +-10 without increasing. I tried to avoid that by using a higher mutation rate in order to add diversity to the population and avoid entering a local maximum, but it resulted that adding a higher mutation rate would just prevent robby from finding a good strategy because it added too much randomness. Lowering the crossover rate usually didn't help finding a strategy faster, besides the one time where it calculated a high fitness value which resulted in the best strategy.

Questions

Why does the best fitness seem to plateau and be bound by an upper limit that is not even close to 500?