COSC343: Assignment 2 report

Guangjie Guo (3060110)
August 30, 2023

1 Introduction

This assignment is to build intelligent agents for cleaners by using Genetic Algorithm. The purpose is to beat random agents as more scores as possible. Each agent can only perceive its immediate surrounding which is a 3*5 grid and its states like the charge left in the battery, the state of its dust bin and the state of success of its last actions.

2 Algorithm

This part is for demonstrating the details of GA I used.

2.1 Agentfunction and Chromosome

The agentfunction I used is quadratic polynomial of all percepts indexed x1 to x63. There are 4*2080 total coefficients in this setting. Therefore, the chromosome will be a vector with 8320 elements. The reason I abandoned the linear model is that the linear model couldn't catch any nonlinear relations, and the problem itself definitely contains nonlinear behaviors. For example, no matter enemy or partner the cleaner encounters, the cleaner with intelligence will not want to collide with it. Because 1 stands for partner and -1 stands for enemy, linear model has no hope to encode this kind of intelligence, and quadratic polynomial can catch this behavior. The reason I didn't consider more complex model, like higher ordered polynomial or neural networks, is that 500 generations evolution might be too few to tune too many parameters into their proper values.

2.2 Fitness function

The fitness function I finally choose is cleaned + emptied + visits. The cleaned stands for total number of dirt loads picked up. The emptied stands for total number of dirt loads emptied at a charge station. The visits stands for total number of squares visited. Visits will encourage the agent to visit more squares. Emptied will encourage the agent visiting charge station more with load but not just simply wandering. Cleaned will encourage the agent to collect more dirts. I have experiment many other combinations, please see the Evaluation section.

2.3 Parents selection

The method for picking parents I use is roulette wheel selection. Firstly, the total fitness for the whole old population is summed up. The probability of selecting any individual as parent is its fitness divided by the total fitness. The advantage of this

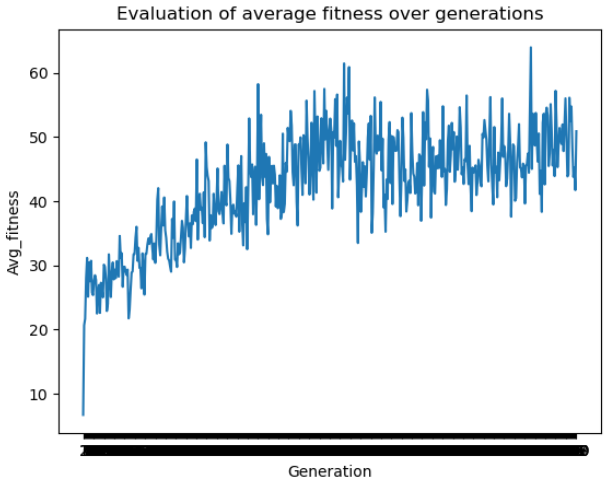

Figure 1: Evaluation of average fitness over generation.

method is good for preserving diversity: every agent, no matter how unlikely, has a chance to reproduce.

2.4 Crossover

The chromosome of parents are just cut in half, and crossover to create one child.

2.5 Elitism and Mutation

The top 15 individuals are passed to the next generation directly as elites. The mutation rate for every element in chromosome of a child is chosen as 0.01. Please see the results and discussion for different settings of elitism and mutation rate in the Evaluation section.

3 Evaluation

In this section, the evaluation of different combinations of fitness, trainingSchedule, elitism, and mutation rate will be presented.

3.1 Elitism and Mutation rate

In this subsection, the trainingSchedule is set as [(random, 500)], which means the whole training is against to random agents. The fitness is set as cleaned + emptied + visits. The number of elites passed to the new generation is represented by the notation "numelits".

Case 1: numelits = 10, mutation rate = 0.01

Please see the Figure 1 for the evaluation of average fitness over generation in this case. The scores for the last 5 games are (220, 193, 189, 243, 208).

Case 2: numelits = 10, mutation rate = 0.005

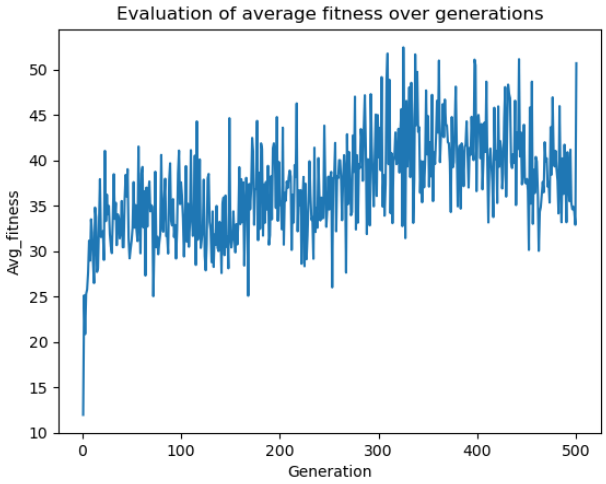

Figure 2: Evaluation of average fitness over generation.

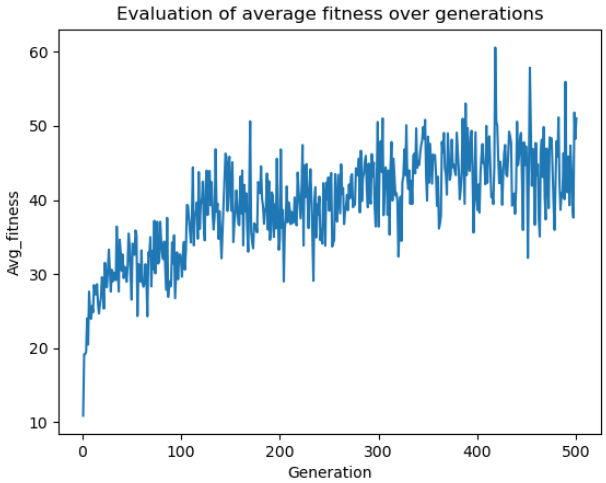

Figure 3: Evaluation of average fitness over generation.

Please see the Figure 2 for the evaluation of average fitness over generation in this case. The scores for the last 5 games are (141, 137, 124, 213, 144).

Case 3: numelits = 10, mutation rate = 0.02

Please see the Figure 3 for the evaluation of average fitness over generation in this case. The scores for the last 5 games are (198, 147, 200, 168, 187).

Case 4: numelits = 15, mutation rate = 0.01

Please see the Figure 4 for the evaluation of average fitness over generation in this case. The scores for the last 5 games are (299, 268, 212, 263, 290).

Case 5: numelits = 12, mutation rate = 0.01

Please see the Figure 5 for the evaluation of average fitness over generation in this case. The scores for the last 5 games are (177, 121, 172, 201, 213).

Case 6: numelits = 20, mutation rate = 0.01

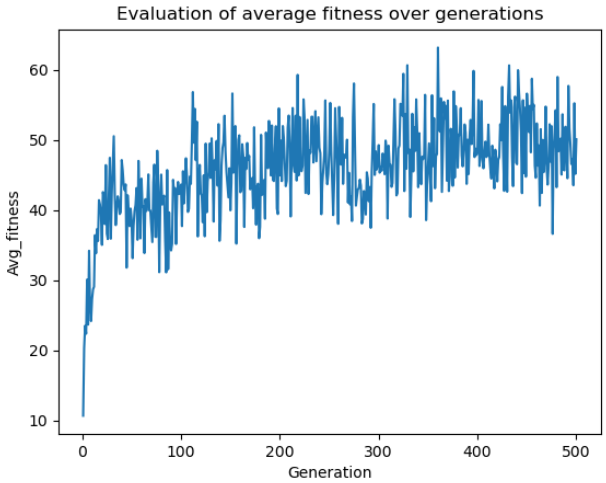

Figure 4: Evaluation of average fitness over generation.

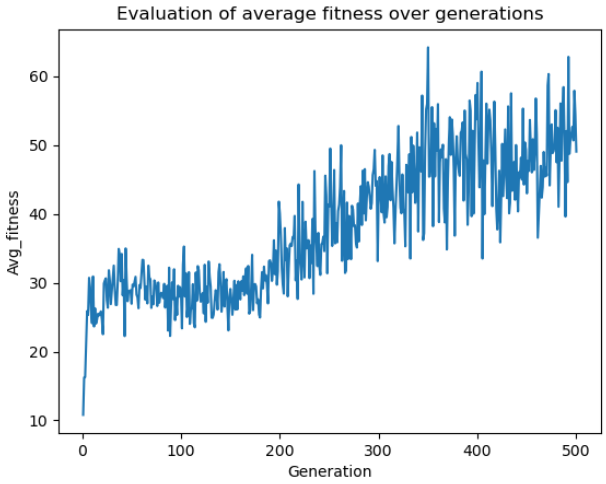

Figure 5: Evaluation of average fitness over generation.

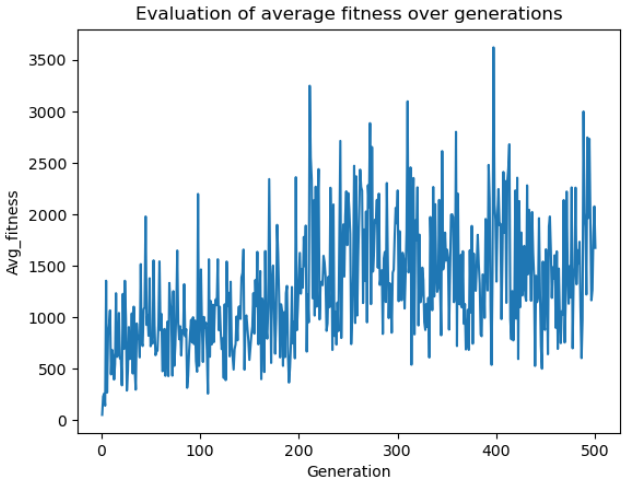

Figure 6: Evaluation of average fitness over generation.

In this case, the scores for the last 5 games are (104, 142, 84, 148, 150).

Comparing the scores of last 5 games above, the best combination of elitism and mutation is numelits = 15, mutation rate = 0.01. According to the evaluation curves of average fitness over generation in these cases, the performance demonstrates vigorous oscillations, but there is an overall trend. One obvious characteristic is that the performance increases rapidly during the first several generations, but the increasing trend gradually slows down after and even a decreasing tail occurs in some cases, like Figure 2 and Figure 4.

3.2 Fitness function

In this subsection, the influence of fitness function selection will be evaluated in different combination. In all cases here, the trainingSchedule is set as [(random, 500)], numelits = 15, and mutation rate = 0.01.

Case 1: fitness = cleaned∗visits∗emptied

In this case, we still use cleaned, visits and emptied but multiplying instead. Please see the Figure 6 for the evaluation of average fitness over generation in this case. The scores for the last 5 games are (134, 140, 123, 143, 94). The performance is worse than its additional version, whose scores can be (299, 268, 212, 263, 290).

Case 2: fitness = (cleaned + emptied + visits)∗successfulactions

In this case, we want to consider the total number of successful actions to avoid collisions. However, the scores at last are (99, 115, 45, 109, 82). It seems like that onsidering successfulactions seems inhibit the total performance in some way.

Case 3: fitness = (cleaned + emptied + visits)∗rechargedenergy

In this case, we want to consider the total recharged energy to encourage the cleaners to staying active more. Please see the Figure 7 for the evaluation of average fitness over generation in this case. The scores for the last 5 games are (167, 208, 140, 242, 252), which is not bad but still couldn't beat the original fitness without multiplying rechargedenergy.

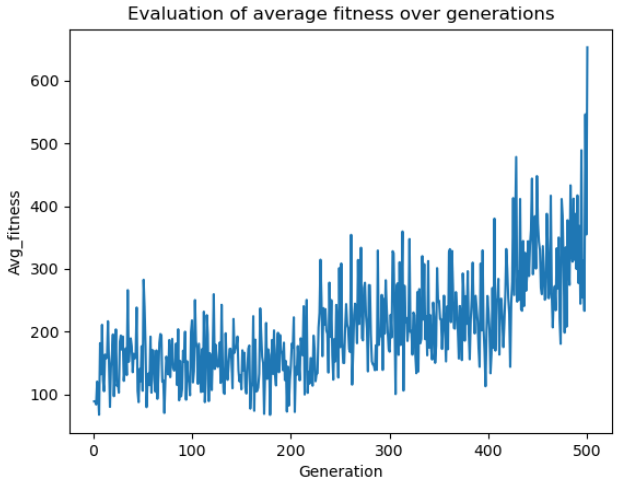

Figure 7: Evaluation of average fitness over generation.

You might notice that I don't just linearly add successful actions or recharged energy to the original fitness. The reason is that successful actions or recharged energy will encourage the cleaner always staying at charge station, and simply added to the fitness will make the performance much worse.

3.3 Training Schedule

Here I tried different training schedules to see if training schedule can influence the evolution process. I think the intelligent self opponents can push the performance better. In all cases of this subsection, we set numelits = 15, and mutation rate = 0.01, and fitness = cleaned + emptied + visits.

Case 1: trainingSchedule = [(random, 100), (self, 100), (random, 100), (self, 100), (random, 100)]

Please see the Figure 8 for the evaluation of average fitness over generation in this case. The scores for the last 5 games are (224, 204, 185, 232, 212).

Case 2: trainingSchedule = [('self', 50), ('random', 50), ('self', 50), ('random', 50), ('self', 50), ('random', 50), ('self', 50), ('random', 50), ('self', 50), ('random', 50)]

Please see the Figure 9 for the evaluation of average fitness over generation in this case. The scores for the last 5 games are (217, 287, 269, 342, 322). This is the best result I have obtained. However, the same setting does not guarantee a same good result. When I trained it the second time with the same setting, the result is much worse than the previous one. Please see the Figure 10 for the evaluation of average fitness this time, and the scores are only (103, 101, 79, 128, 109). Comparing Figure 10 with Figure 9, you can see the cleaners didn't learn much this time. I think the reason is the stochastic nature of Genetic Algorithm.

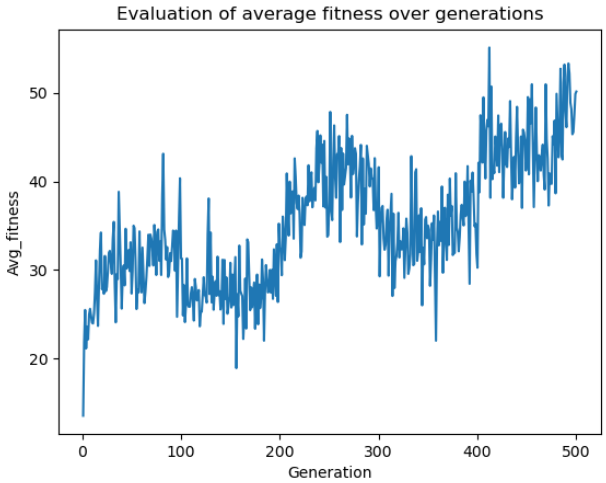

Figure 8: Evaluation of average fitness over generation.

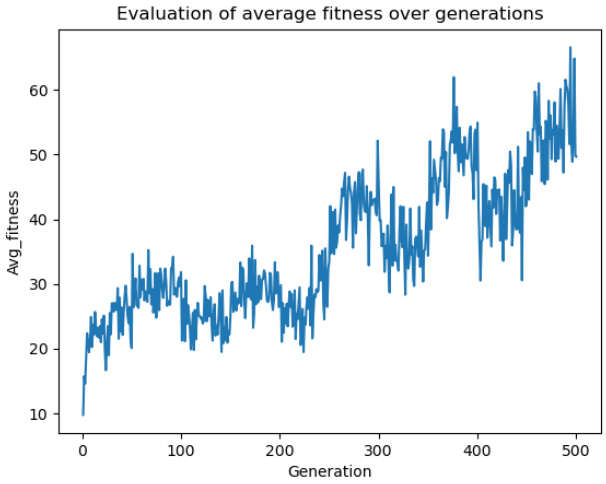

Figure 9: Evaluation of average fitness over generation.

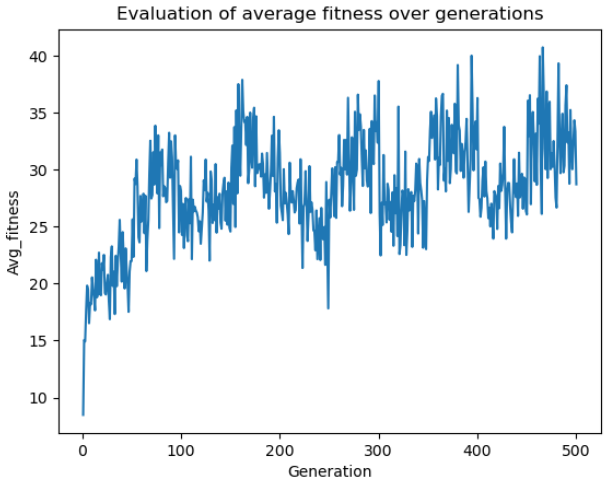

Figure 10: Evaluation of average fitness over generation on the same setting for Figure 9.

4 Conclusion

In conclusion, I use the Genetic Algorithm to tune a quadratic polynomial model as the agent function for cleaners. Indeed, it is an art to set the hyper-parameters, like elitism, mutation rate, fitness function and training schedule. In addition, the stochastic nature of GA makes it even more difficult to find the optimal hyper-parameters setting under a limited evolutionary process. The setting of my best result is: numelits = 15, and mutation rate = 0.01, fitness = cleaned + emptied + visits, trainingSchedule = [('self', 50), ('random', 50), ('self', 50), ('random', 50), ('self', 50), ('random', 50), ('self', 50), ('random', 50), ('self', 50), ('random', 50)]. For seed = 2, the scores of last 5 games can be (217, 287, 269, 342, 322).