Adaptive and Learning Agents (ALAS) Part B Open Assessment

Y1403115

The prisoners dilemma is probably the most famous game theoretic problem. A lot of active research has gone into analysing evolutionary approaches for the repeated version of the problem [1] [2] [3] [4]. In this project a multi agent evolutionary algorithm is used to produce intelligent behaviour for a modified version of the repeated prisoners dilemma with societies.

1 Agent Behaviour Representation

In the provided implementation, each agent stores it's cumulative score along with the society it currently belongs to. Societies are encoded as integers 0 - green, 1 - red, 2 - blue and 3 - brown.

The internal representation of each agent's behaviour (genotype or also called strategy in this project) follows the one used in [5]. It is a list of 32 integers. The list elements are integers in the range $[-1; 3]$, each corresponding to a new society with the additional -1 - remain in the same society. After each played round an agents decides which action to take by considering three factors - their old society, their opponent's society and weather or not they have a higher cumulative score. These three factors are combined into an index (see Equation 1), which is used to determine which of the internal list elements is going to be chosen as the new society.

$$16 \times \mathit{isScoreBigger} + 4 \times \mathit{society} + \mathit{opponentSociety} \tag{1}$$

2 Game Implementation

The game environment is implemented in *game.py.* It consists of some utility functions for calculating the payoff of agents, the *Agent* class and a *Game* class.

As discussed in section 1, agents have a cumulative score, society and strategy. Their society is assigned randomly upon creation. They also have a *play* method, in which they calculate their payoff from the encounter with a particular opponent, and a *changeSociety* method in which they choose what action to take depending on the outcome of the encounter and the societies they and their opponents belong to.

The *Game* class stores a list of agents, participating in the current game and the number of iterations N, the game lasts. It has a single *play* method which runs a single game of N encounters. In the *play* method, first all agents' scores are cleared, after which N encounters of randomly chosen agents are played out. In a single encounter two agents are chosen randomly, after which they play the prisoners dilemma game, receiving payoffs, according to their current societies. At the end of the encounter they are allowed to change societies according to their strategies(genotypes).

In this environment, a single run of the game (N encounters) is used to estimate the scores of all agents. The score each agent obtains is used as a fitness value for the evolutionary algorithm described in section 3. Although the choice of players is random for each encounter, for a sufficiently large N, the scores they achieve would be a good estimate of their true fitness. However, this random element also means that the number of encounters and agent plays is highly dependent on the population size. This means that it is not feasible to compare experiments with different population sizes directly.

3 Evolutionary Algorithm

An evolutionary algorithm was used to produce intelligent behaviour in agents. The algorithm parameters are summarised in Table 1 and were chosen empirically and based on available computational resources. A clear separation between genotype and phenotype can be observed in the implementation, as the algorithm evolves only the strategy attribute of agents, as opposed to the agents themselves. Tournament selection with a small tournament size was chosen to increase selection pressure. The fitness evaluation of the whole population is done in a single repeated prisoners dilemma game. As discussed in section 2, the random element of the evaluation procedure reduces computational demand and allows a relatively high number of generations. However, in order to ensure that the whole population participates in a reasonable number of games(fitness evaluations), the population is kept quite low.

Individual representation (genotype)	List (see section 1)
Selection Tournament size	Tournament selection 3
Crossover Crossover probability	Two point crossover 0.5
Mutation Mutation probability	Replace attributes with uniformly drawn integers 0.2
Fitness function N	Repeated prisoners dilema game (see section 2) 1000
Number of generations	10000
Population size	20, 100

Table 1: Evolutionary algorithm parameters

Two versions of the evolutionary algorithm were implemented and compared - one where all agents evolve and on in which one agent evolves at a time.

Multi Agent Evolution

In the multi agent evolution procedure all agents evolve at the same time. After selection, all genotypes participate in crossover and mutation. After which, their fitnesses are evaluated and they form the population in the next generation.

Single Agent Evolution

In the single agent evolution case, only one agent is allowed to adapt at a time. Each generation, the adapting genotype is chosen at random from the population. In order to allow for the chosen agent to participate in crossover, selection is performed on the whole population as usual. In the crossover and mutation phases, only the selected individual is allowed to take part. After that, the adapted individual is inserted into the old population and fitnesses are evaluated on the whole population.

4 Experiments and Evaluation

In order to compare the performances of agents produced by the evolutionary algorithm from section 3, an evaluation procedure was developed. In the evaluation procedure the best individual produced from evolution is played against populations of agents who never change societies. In order to reduce random error, the agent plays 1000 games (each of N encounters), each within a newly generated population, with random societies assigned, but who never change their societies. The score and rank (position relative to the other agents in terms of score) of the best individual are stored and used to compare performance with other runs.

A baseline agent, who never switches societies was also evaluated using the same procedure. However, the baseline agent was assigned a new society for every new game.

A total of 6 experimental runs were performed with the different algorithm variations and different population sizes (see Table 2). The results are discussed in section 5.

Baseline with population=20	Multi Agent Evolution with population=20	Single Agent Evolution with population=20
Baseline with population=100	Multi Agent Evolution with population=100	Single Agent Evolution with population=100

Table 2: Experimental runs

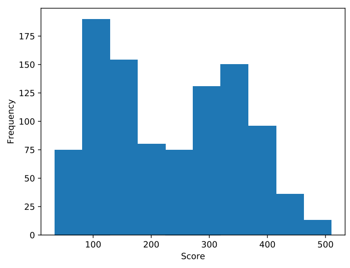
5 Results and Conclusion

In this section, the results from the experiments mentioned in section 4 are discussed. For each of the six experiments, the final score and rank of the evaluated agent was were recorded. Statistical tests were performed for all the results in order to determine their significance. The score results will be discussed first, followed by the rank results.

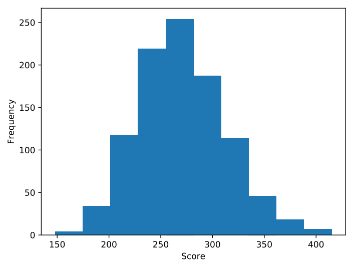
The distributions of the scores from all 1000 runs for each algorithm variation with population size 20 are shown on Figure 1. The two evolutionary strategies follow a more or less normal distribution, as opposed to the baseline model. It is clear that the Multi Agent Evolution algorithm outperforms the other two, with a mean score between 250 and 300.

In the experiment with population 100, the scores are much lower (Figure 2), due to the fact that each agent statistically plays less games. Unlike the results from the population 20 case (Figure 1), there does not seem to be much of a difference between the two evolutionary algorithms. Both evolutionary algorithms seem better than the baseline.

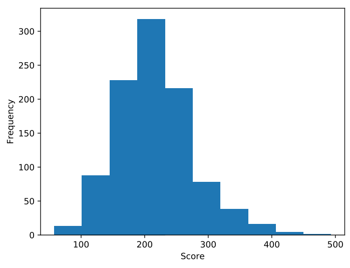
A Kruskal-Wallis H test was performed on all score results, in order to determine if there is a significant difference between the different groups. This is a non-parametric test with



(a) Baseline 20

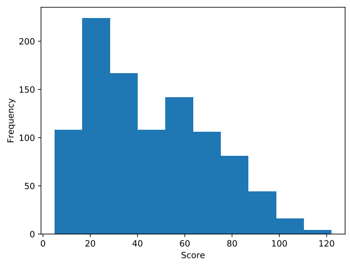


(b) Multi Evolution 20

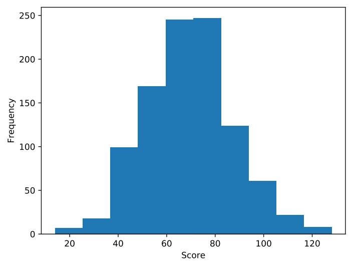


(c) Single Evolution 20

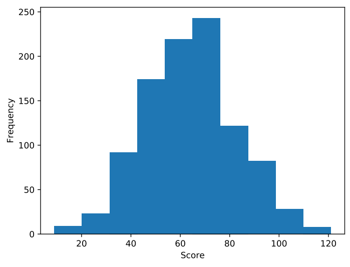
Figure 1: Score distributions for group ran with population=20



(a) Baseline 100



(b) Multi Evolution 100



(c) Single Evolution 100

Figure 2: Score distributions for group ran with population=100

a null hypothesis that all samples come from the same distribution. The Kruskal-Wallis test was chosen over the ANOVA because the distribution of the baseline experiment is obviously not normal and does not satisfy the assumptions made by the ANOVA test. The test results are shown in Table 3. The p-values for both the population 20 and population 100 experiments are much lower than the 0.05 significance level.

	population = 20	population = 100
statistic	330.489	530.617
p-value	1.718e-72	5.995e-116

Table 3: Kruskal-Wallis H test for scores

The Kruskal-Wallis test only tells us that the distributions of all samples are not the same but it does not tell us which ones are different and which ones are not. To determine the significance of differences between pairs of experimental groups, a post-hoc test needs to be used - Dunn's test. This is also a non parametric test. The results from Dunn's test for the population 20 scores are shown in Table 4 and they confirm that the differences between each pair of samples is significant with very small p-values.

	Baseline-Multi	Baseline-Single	Multi-Single
statistic	-11.731	6.161	17.892
p-value	8.830e-32	7.200e-10	0.0

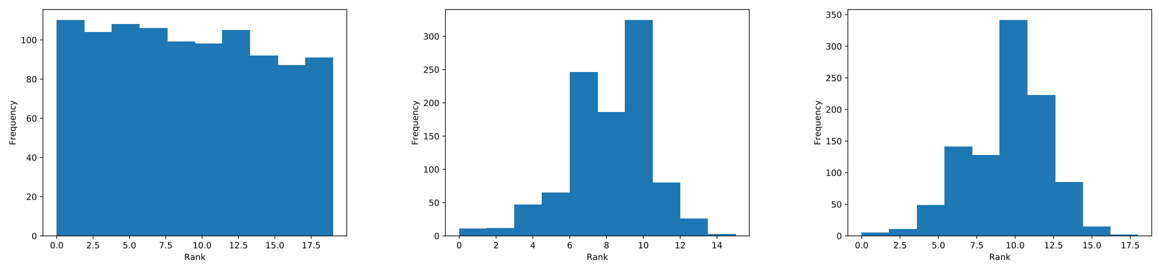
Table 4: Dunn's test for scores population=20

The result from Dunn's test (Table 5) for the scores of the two evolutionary algorithms with population 100, however show that the two samples come from the same distribution. Nevertheless, they are both significantly different from the baseline.

	Baseline-Multi	Baseline-Single	Multi-Single
statistic	-22.121	-16.622	5.498
p-value	1.953e-108	4.765e-62	3.821

Table 5: Dunn's test for scores population=100

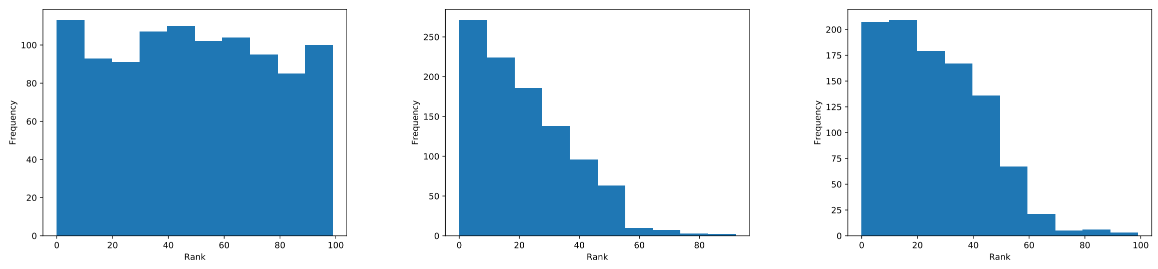
Following the same analysis for the ranks achieved by the three versions of the algorithm, we see that the distribution for the baseline is almost uniform in both the population 20 and 100 cases (Figure 3 and Figure 4). Looking at the distributions for the multi and single agent evolution algorithms in the population 20 experiment, it seems like the multi evolution algorithm is better.



(a) Baseline 20 (b) Multi Evolution 20 (c) Single Evolution 20

Figure 3: Rank distributions for group ran with population=20

In the population 100 case, however the multi agent evolution algorithm seems to outperform the single agent evolution(Figure 4).



(a) Baseline 100 (b) Multi Evolution 100 (c) Single Evolution 100

Figure 4: Rank distributions for group ran with population=100

Again, performing the Kruskal-Wallis test and subsequently Dunn's test, we see that the results for the rank distribution are all significant (Table 6, Table 7 and Table 8).

	population = 20	population = 100
statistic	117.282	521.604
p-value	3.408e-26	5.432e-114

Table 6: Kruskal-Wallis H test for ranks

	Baseline-Multi	Baseline-Single	Multi-Single
statistic	-6.520	-4.227	-10.748
p-value	7.000e-11	2.359e-05	6.024e-27

Table 7: Dunn's test for ranks population=20

	Baseline-Multi	Baseline-Single	Multi-Single
statistic	21.878	16.613	-5.265
p-value	0.0	0.0	1.398e-07

Table 8: Dunn's test for ranks population=100

After comparing the two evolutionary approaches with the baseline, it is clear that they are superior. In the smaller population experiment the Multi Agent Evolution algorithm definitely outperforms the other one judging by both the score and rank achieved. In bigger population experiment it is harder to say which one of the two is better, although there is some evidence that the Multi Agent Evolution achieves a higher rank overall.

This project presented a multi agent evolutionary approach to a variation of the repeated prisoners dilemma. Two versions of an evolutionary algorithm were compared and evaluated. Further experiments can be performed with greater population sizes and number of generations. Furthermore, the mutation and crossover rate can fine tuned to achieve maximum performance.

References

[1] R. Boyd and J. P. Lorberbaum, "No pure strategy is evolutionarily stable in the repeated Prisoner's Dilemma game," en, *Nature*, vol. 327, no. 6117, pp. 58–59, May 1987, 00565, ISSN: 1476-4687. DOI: 10.1038/327058a0. [Online]. Available: https://www.nature.com/articles/327058a0 (visited on 05/09/2018).

[2] R. Boyd, "Mistakes allow evolutionary stability in the repeated prisoner's dilemma game," *Journal of Theoretical Biology*, vol. 136, no. 1, pp. 47–56, Jan. 1989, 00234, ISSN: 0022-5193. DOI: 10.1016/S0022-5193(89)80188-2. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022519389801882 (visited on 05/09/2018).

[3] J. H. Miller, "The coevolution of automata in the repeated Prisoner's Dilemma," *Journal of Economic Behavior & Organization*, vol. 29, no. 1, pp. 87–112, Jan. 1996, 00482, ISSN: 0167-2681. DOI: 10.1016/0167-2681(95)00052-6. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0167268195000526 (visited on 05/09/2018).

[4] J. Farrell and R. Ware, "Evolutionary stability in the repeated prisoner's dilemma," *Theoretical Population Biology*, vol. 36, no. 2, pp. 161–166, Oct. 1989, 00107, ISSN: 0040-5809. DOI: 10.1016/0040-5809(89)90027-0. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0040580989900270 (visited on 05/09/2018).

[5] D. Eck, *Genetic Algorithms Demo in JavaScript*, 00000. [Online]. Available: http://math.hws.edu/eck/jsdemo/jsGeneticAlgorithm.html (visited on 05/08/2018).