

Submitted in part fulfilment for the degree of BEng

Applying Coevolutionary techniques to Competitive self-play Reinforcement Learning to better solve non-stationary problems

Tom Clark

Click or tap to enter a date.

Supervisor: Dan Franks

Acknowledgements

Click or tap here to enter text.

Click or tap here to enter text.

Table of Contents

Table of Contents

[Executive Summary 6](#_Toc99962812)

[1 Introduction 1](#_Toc99962813)

[2 Literature Review 2](#_Toc99962814)

[3 Methodology 4](#_Toc99962815)

[3.1 The Minimal Substrate 4](#_Toc99962816)

[3.2 Experimental Setup 5](#_Toc99962817)

[4 Experiments and Results 9](#_Toc99962818)

[4.1 Experiment 1 – Loss of gradient 9](#_Toc99962819)

[5 Conclusion 12](#_Toc99962820)

[Appendix A 13](#_Toc99962821)

[Appendix B 14](#_Toc99962822)

[Bibliography 15](#_Toc99962823)

Table of Figures

Table of Tables

[**Table 1.1**: A table with its caption. 2](#_Toc525300873)

# Executive Summary

# Introduction

There are two other types of Machine Learning, unsupervised and supervised learning. Unsupervised learning uses data which hasn’t yet been classified and first must be clustered and categorised, whereas supervised learning uses data which has already been classified by a human. With respect to RL, these two concepts can be used in the “bootstrapping” phase of RL, when the original agents are set up. For example, the first agents may learn from supervised and curated examples and then transition to RL.[11]

In my research of Reinforcement Learning and Coevolutionary literature, only 4 of the 21 papers mentioned both types of machine learning [12] [13], and of those only two applied coevolutionary principles to Reinforcement Learning. [9], [14] Little research has been made into the vast amount of research into coevolutionary learning and applying it to much younger field of Reinforcement Learning. This is what this paper will attempt to examine.

One of the RL journals I read used League training, a MARL algorithm “designed to address the cycles commonly encountered during self-play training” by creating a Hall of Fame, a collection of previous agents from past generations to compete against the current generation, thereby reducing the chance of a cycle.[11] This problem has also been investigated and solved in Coevolution, providing evidence that some of the problems in Coevolution are also present in RL. However, no reference was made to research into Coevolutionary algorithms and their solution to the problem in that journal, implying that the fields of research are operating in parallel to each other.

# Literature Review

Reinforcement Learning (RL) is a method of machine learning that maximises an agent’s Objective Function performance within an environment, by learning the optimal parameters for the agent within that environment.[1] The environment includes everything outside the agent. A RL environment is usually presented in the form of a Markov Decision Process (MDP). A MDP takes 5 inputs; the finite state space S, the finite action space A, the transition function T : S x A x S -> [0,1], the reward function R : S x A x S -> R and the discount factor γ ∈ [0,1].[2] An MDP attempts to find a policy π, a function that fully defines the action of an agent whilst in a specific state, that maximises the reward/fitness function.

However, most MDPs used in RL find a stationary policy, a policy that is independent of time. This is a key weakness when they are applied to a non-stationary environment; an environment which changes over time, because now the expected value for each stationary policy can vary.[3] One such example of a non-stationary problem is the simple two player game of Rock-Paper-Scissors, played in a repeated format. Here, two agents will play against one another and attempt to maximise the number of times they win. An ideal policy would learn their opponent’s preference and attempt to counter that by changing its own policy, which prompts the opponent to adjust accordingly, introducing non-stationarity due to changing behaviours.[4]

One such area that where non-stationarity might be introduced into problems is in competitive multi-agent environments. In a single agent environment, there is only a single agent operating in that environment, whereas in a multi-agent environment, multiple agents may be operating and competing within the environment. A popular method of training agents in a competitive multi-agent environment is through competitive self-play RL.

Competitive self-play RL is a type of multi agent RL (MARL) that has an agent play against another agent or agents, or a copy of itself, and judging its fitness off its performances in the matches. The environment for each “match” would be the opposing agent or agents and any other variables[5]. As described above, non-stationarity is evident as agents build up a type of “memory” as they play against other agents and attempt one-up each other.

There exists a separate and much older area of machine learning known as Competitive Coevolution, hereby referred to as Coevolution, that has remained largely ignored by RL research. Coevolution is a type of Evolutionary Algorithm (EA) which was first introduced in 1990 [11] that tasks individuals within a population of agents to compete against each other, thereby measuring their performance with respect to the rest of the population.[12] As in other EAs, agents from the population will then be selected, mutated and cross-bred to produce new “offspring” agents for the next generation. The general aim of Coevolution is to create an “arms-race”, whereby agents in a population will progressively one-up the others, producing an adaptive gradient that is difficult to engineer manually.[13]

The similarities between Coevolutionary algorithms and competitive self-play RL are self-evident; each creates an optimal agent through iterative generations and evaluates agents by having them compete against other agents. Coevolution however holds an innate advantage over RL in non-stationary environments as Evolutionary Strategies do not rely on stationary transition probabilities, whereas MDP-based RL methods do. [9]

Research has also been done into the co-application of MARL and Coevolution with a view to create scalable learning techniques.[9] The paper used Atari games as a benchmark for their Coevolutionary Algorithms and compared them against a Deep-Q Network (DQN) variant which learns with RL. Their results showed that the Coevolutionary algorithms can be superior to the DQN on several of the games, and hypothesised that the multi-agent aspect of the problems introduced non-stationarity, which as mentioned above is no issue for Evolutionary Strategies but can cause problems for MDP based solutions like DQNs.

However, the method of Coevolution introduces 3 well researched and largely solved problems. The first is Cycling, whereby the agents may cycle over and over through a set of “optimal” agent types which lack transitivity, e.g. A<B<C, but C<A, leading to a cycle over these 3 types without any progress. Another problem is a loss of gradient called Disengagement where one set of the population may dominate all others, thereby rendering it impossible for any meaningful change to be measured. Lastly is a loss of focus, where agents may focus entirely on an opponent’s weakness thereby leading to agents which don’t play to win the game, but to win against that specific opponent, creating weak, non-adaptive agents when challenged by other opponents.[14]

There lies the motivation for this paper; can Coevolutionary techniques be applied to RL to better solve non-stationary problems, does the combination succumb to the same problems that Coevolutionary algorithmic learning experiences and do the existing solutions for Coevolution solve any of those problems.

# Methodology

## The Minimal Substrate

In this section I will outline the minimal substrate that will be used in the experiments. This minimal substrate is based on Richard A. Watson and Jordan B. Pollack’s minimal substrate.[14]

The goal of each agent is to maximise a, a scalar value it possesses. This maximisation however will use a fitness function, that returns a value for ’s fitness with respect to a set of other numbers, . contains many other agents all with their own value. I will let count the number of members of that beats:

Eq1

Where = 1 if , 0 otherwise.

Therefore, instead of a trivially easy game of maximising by itself, will be evaluated and increased with respect to all other agents, which will also be increasing.

The second component of the minimal substrate is a second dimension, such that each agent has two scalar values on two dimensions, and . This second dimension is an abstraction of the strengths of each agent, as in more complex implementations of machine learning, agents may possess different strengths and weaknesses. The introduction of this second dimension means that we cannot reduce both agents to a single dimension and compare them, e.g. , as for some opponents, the value of dimension  may be more important that dimension .

As with the original minimal substrate, I will be comparing two agents with regards to whichever dimension is furthest apart.

Eq2:

where

Overall, the goal of each agent is to maximise all its dimensions. There exist, however, potential pitfalls for agents to fall into, such as over-specialising on a single dimension whilst the other remains small, in order to maximise its subjective fitness; its fitness compared to all other agents in the environment. This effect appears in Coevolution, and part of the experiment is to see if it manifests in RL as well.

The original minimal substrate paper also includes a third equation to add intransitive superiority. That is, that in a random game with 3 agents, A, B and C, agent A beats B, B beats C, but C beats A. This creates the scenario where A’s superiority over B doesn’t intransitive, and doesn’t automatically allow it to beat C. This is a key concept in coevolution and can result in algorithmic failure as agents cycle over and over.[15] As with the original minimal substrate, I will be modifying Equation 2 so that the dimension compared between 2 agents will be the dimensions in which the players are closest.

Eq3:

where

I will use the original minimal substrate paper’s example to demonstrate the intransitive superiority introduced by Equation 3 here with 3 agents, a=(1, 6), b=(4, 5), c=(2, 4). a beats b as they are closest in the y dimension, and , b beats c as they are closest in the y dimension and , but c beats a as they are closest in the x dimension and . Overall however, the optimal agent is still one which is maximal in both dimensions and beats all other agents.

## Experimental Setup

Whilst I shall use the previous equations as defined above, there are several representation and RL setup decisions to be made.

I will be representing the Environment with a custom class, which contains a list of N agents, where the agents are made from another custom-built class. The agents have 2 attributes to store their dimensions, first and second, which store scalars. When the Environment class has its reset method called, it initialises the N length array of agents, and initialises their dimensions as random integers between 0 and 10.

The Environment class contains methods to evaluate the agents using either Equation 1, 2 or 3 and calculate their reward. When evaluating agents, if the dimensions between two agents are equidistant, I will default to comparing dimension y. If two agents compare a dimension, and find it is equal, neither will get a bonus to their overall reward. This is to eliminate any possibility of emergent cooperative behaviour.

The Environment class also makes use of OpenAI’s gym module’s Discrete class for representing the observation space and action space. I’ve chosen to represent the observation space, the number of states an agent can be in, as a set of 6 possible states. First, the Environment class calculates how many other agents are greater than the evaluated agent in dimension X, how many are equal in dimension X, less than in dimension X, greater than in dimension Y, equal to in dimension Y and less than in dimension Y. The state is then the index of whichever one of these numbers is greatest: e.g. if most less than the agent in dimension Y, the state will be 2.

There are two possible directions to take the action space, the set of actions available to agents. Design A uses the original minimal substrate paper’s scalar representation; the scalar in each dimension was represented as a “fixed length binary string and the value they represent is given by the unitation of the string” where the string length was 100. This choice was so that there was a biased mutation, i.e. instead of simply using an integer representation, where mutation may only increase or decrease by a set amount, with no bias either way, whereas by using a binary string representation gives a mutation bias; a string with half 0’s or more is more likely to increase, and a string with half 1’s or more is more likely to decrease. This mutation bias is important in real applications of more complex problems and shall be used for this design. Design A also has 3 possible actions: flip a bit at a random index in dimension X, flip a bit at a random index in dimension Y or take no action. This design is very similar to the coevolutionary representation in the original Minimal Substrate model and represents individuals mutating randomly but doesn’t consider the nuances of RL.

Design B uses regular integer scalars, with a maximum value of 100 and minimum of 0, and has 4 possible actions: increase X by a scalar Z, decrease X by Z, increase Y by Z, decrease Y by Z. Z is a random integer between 1 and Z max, which I have set to 3, and attempting to increase X or Y over 100, or decrease it below 0 just sets it to 100 or 0 respectively, and counts as “skipping” an action. This design better accommodates RL’s policy-based strategy rather than selecting based off of random mutations.

Both these designs use a discrete action space with a pre-defined set of actions, instead of a continuous action space where actions are real valued vectors. A continuous action space implementation would possibly be an even better representation and allow X and Y to change by uncapped amounts, but I have chosen not to take this approach as it will add extra complexity to the whole experiment without providing any extra exploratory value to the experiment, and it would possibly add an extra computational burden can’t be afforded on the hardware I am using. In conclusion, I will only be using Design B. I made this decision because the random nature of evolution doesn’t translate well into policy making; an individual choosing to mutate its X dimension isn’t making a choice about whether it wants to increase or decrease that dimension, only that it wants to change it. Design B on the other hand provides this choice. I also have created an extension of Design B that is called Design C. Design C is exactly the same, except has 2 populations of individuals and 2 q tables. In Design C, all agents compete against each other, but agents from population 1 share and update q table 1, and also get their actions from it as well. This Design will be used to examine loss of gradient, so it can be better compared to the original paper’s plots.

The Environment contains a step method, that is given a list of N actions in the form of a list of integers, with the ith action being intended for the ith agent. The Environment applies each action to the intended agent, then calculates the rewards with the chosen equation, which is stored in an N length list, so that the reward for the ith agent is the ith item in the rewards list. The same is done for the observations for each agent; they are calculated and stored in an N length list. The Environment then increments a current generation counter, which keeps track of how many times step has been called since the Environment was last reset. If the current generation counter equals the max generation variable, which I have set to 100, then the Environment sets its done flag to True. After the step method is complete, it will return the list of observations, rewards and the done flag. When the reset method is called, it will calculate and return an observation list as well; this is the list of initial states for each Agent before any actions have been taken.

I decided upon a decentralised Q-table approach for the RL part of the program, decentralised meaning that each agent has a separate Q-table. The Q-table calculates and keeps track of the expected rewards for actions at each state, so that the overall policy, the strategy, is improved over time.

To implement this, I created a list of length N, which contains N Q-tables, where the ith Q-table belongs to the ith agent. Each Q-table is initialised as a 2d array full of 0’s, with a shape of (6, 4), 6 being the observation space, and 4 being the action space. For the ith agent, the Q-value for action A whilst in state S will be at index [i][S][A]. This is all contained within a Q Network class I created and done at initialisation. I also chose hyperparameters of: 0.1 for the learning rate α, 0.6 for the discount rate γ and 0.2 for the greed value ε.

The Q Network then runs for M iterations. At the start of each iteration it resets the environment and sets the current state to the returned observation. The environment then loops through until the max generation is reached. At each cycle of the loop, a set of N actions is decided for each agent. There is a random ε chance that a random action is chosen, otherwise, so long as the Q values are not all 0 at index [i][S] it will select the index with the highest Q value, with that index corresponding to the chosen action, A. If the Q values are all 0 it picks a random action. This prevents the network having a very strong natural bias to whatever action corresponds to index 0, as a quirk of choosing the maximum Q value is that it defaults to index 0 if all Q values are equal.

The actions are then passed to the Environment’s step method, which returns the observations and rewards, as well as the done flag. The Q values for each agent is then updated using the formula

Where

And returns the index of the maximum Q value of the list of Q values at index [i][observation] in the Q table, for the ith agent. The state is then set to the observation, which will carry over for the next cycle.

My implementation and source code can be accessed on Github[[1]](#footnote-1).

# Experiments and Results

First, an explanation for how plots were generated and how to read them.

Figure 1 represents a list generated from Design B during training runtime. This list shows that action 0 was taken w times, action 1 x times, action 2 y times and action 3 z times, where w, x, y and z sum up every time that action was performed by any agent over all generations and all iterations.



Figure - Number of times each action was taken

Figure 2 shows the plots generated after training has taken place. The top row plots the average subjective rewards of the individuals against the generations, and the bottom row plots the average of the 2 scalar values against the generations. The plots are all colour coded and colour synced; in the top left, the green line shows the average reward for individual Z, and the plot below it has two green lines that show the average scalar value for dimension X and Y respectively. Each graph shows 5 individuals to make it easier to read.

In Design B, the values on the Y axis that are plotted are the average for each individual over all iterations. During the training, the subjective rewards and scalar values of every individual are stored, along with the generation counter. To calculate the average subjective reward, the plot then sums up individual Y’s subjective reward over all iterations when the generation counter equals 0, then divides by the number of iterations, which generates the average reward at generation 0, this is then repeated for every other generation for individual Y, then for every other individual. The same is done to calculate the average scalar value.

Diagram

Description automatically generated

Figure - Plots of Rewards and Scalars for Design B

In Design C the plots are different, as show in Figure 3. The two lines represent the average of all individuals from each population. The red line would represent population 1, and the green line population 2.

The Y axis values are calculated from the average subjective reward and average scalar value for each population, which in turn is the average of all the average subjective rewards and scalars from every individual belonging to that population.

Chart, line chart

Description automatically generated

Figure - a Design C plot where agents have only one scalar value

## Experiment 1 – Loss of gradient

These experiments were all performed with Equation 1, and thus agents have only a single scalar value. The purpose of this experiment is to show that using the subjective fitness does not necessarily result in the creating the best individuals with a high objective performance, i.e. the highest scalar value.

First, I trained Design B with 25 individuals over 50 iterations.



Figure

Chart, diagram

Description automatically generated

Figure – all iterations

Chart

Description automatically generated

Figure 6 – final iteration only

Figure 4 shows that action 0, increasing the scalar value, was the preferred action at the end of training which is what was expected. Figure 5 shows that the average subjective rewards of all individuals eventually decreased to around ~8 or ~6, even though the average scalar value for all individuals climbed to over 80 for all individuals except one. After generation ~80 most individuals reached their highest average scalar value, and after this point the subjective fitness didn’t vary significantly for any agent.

Figure 6 shows only the final iteration. 4 individuals had their subjective fitness fall to 0, or to far below the other individuals in the environment. In turn, these individuals’ scalar values all flattened out and didn’t increase again significantly.

Next, I trained Design C for 50 iterations, using 2 populations which have 25 individuals each.



Figure

Chart, line chart

Description automatically generated

Figure

Figure 7 shows that action 0 was again the preferred action. Figure 8 again shows that after the average scalar value hit its maximum at around generation 80, the subjective reward did not vary significantly.

Finally, I ran Design B again for 50 iterations but with only 2 individuals.



Figure

Chart, line chart

Description automatically generated

Figure

Figure 9 again shows action 0 was the preferred action. Figure 10 shows a different result, here the teal individual dominated the red individual, and teal climbed to ~80 for its scalar value whereas red maxed at ~40. The subjective rewards graph also shows that the reward for teal was 1 – the reward for red, and vice versa. Taking this plot at face-value gives the wrong conclusion however, and instead a plot of a single iteration is required.

Chart, histogram

Description automatically generated

Figure 11

Chart, line chart

Description automatically generated

Figure 12

Figure 11 shows the penultimate iteration, and Figure 12 shows the final iteration. From these two plots, the true conclusion can be found: whichever individual started with the higher initialisation ended up dominating the other. This is because there is no subjective reward gradient in this 2-agent environment, either the individual is winning or it is losing; there is no way to tell how much it is winning by or losing by, so therefore the losing individuals never learn how to come back from losing, whilst winning individuals continue to reward increasing their scalar value.

To conclude Experiment 1, a loss of gradient has been shown. In Figure 5 and 8 this is shown by the subjective fitness of individuals not improving significantly after a certain point, this is because even though the individuals themselves have a high objective fitness, their scalar value, they are not any better in comparison to the other agents, as all other agents have also improved proportionally. This is known as the Red Queen Effect, and has been previously documented in Coevolutionary research[15] but to my knowledge has not yet been documented in RL.

## Experiment 2 – Focusing

In these experiments I will be using Equation 2, and all agents will have 2 dimensions.

Design B 25 agents 50 iters



Figure 13

Chart

Description automatically generated

Figure 14 - Objective fitness and Rewards

Notice the objective fitness fails to reach 200, the maximum value possible

Design B 25 agents 50 iters, 10 dims 50=max value



Chart, diagram

Description automatically generated

# Conclusion

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Pellentesque

# Appendix A

Delete as necessary.

Fusce congue mauris leo, nec sagittis justo convallis quis. Maecenas varius tellus vel nulla facilisis accumsan. Nunc in facilisis nulla. Suspendisse ac molestie ante. Duis vulputate sit amet tellus eget efficitur. Cras ut dolor at dolor molestie dignissim. Proin eu massa dapibus, feugiat quam auctor, porttitor urna. Integer porttitor, libero a egestas lacinia, neque leo luctus elit, id iaculis leo quam porta est. Integer ut porttitor tortor.

# Appendix B

Donec felis sem, consectetur eu cursus ac, auctor sollicitudin velit. Sed hendrerit, metus et gravida vulputate, urna lacus facilisis neque, sed accumsan tellus nibh vel metus. Ut tempus felis neque, vitae rutrum ligula tristique sit amet. Morbi nec dui lectus. Ut fringilla orci in faucibus sodales. Integer dolor arcu, pellentesque a blandit non, volutpat vitae sem. Curabitur sit amet nunc tincidunt, mollis purus in, luctus eros.

# Bibliography

[1] O. Vinyals *et al.*, “Grandmaster level in StarCraft II using multi-agent reinforcement learning,” *Nature*, vol. 575, no. 7782, pp. 350–354, Nov. 2019.

[2] T. P. Runarsson and S. M. Lucas, “Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board go,” *IEEE Trans. Evol. Comput.*, vol. 9, no. 6, pp. 628–640, Dec. 2005.

[3] K. Chellapilla and D. B. Fogel, “Evolving neural networks to play checkers without relying on expert knowledge,” *IEEE Trans. Neural Netw.*, vol. 10, no. 6, pp. 1382–1391, 1999.

[4] A. Kianercy and A. Galstyan, “Coevolutionary networks of reinforcement-learning agents,” *Phys. Rev. E Stat. Nonlin. Soft Matter Phys.*, vol. 88, no. 1, p. 012815, Jul. 2013.

[5] D. Klijn and A. E. Eiben, “A coevolutionary approach to deep multi-agent reinforcement learning,” *arXiv [cs.NE]*, 12-Apr-2021.

[6] J. Oh *et al.*, “Discovering Reinforcement Learning Algorithms,” *arXiv [cs.LG]*, 17-Jul-2020.

[7] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, “An Introduction to Deep Reinforcement Learning,” *arXiv [cs.LG]*, 30-Nov-2018.

[8] E. Lecarpentier and E. Rachelson, “Non-Stationary Markov Decision Processes a Worst-Case Approach using Model-Based Reinforcement Learning.”

[9] G. Papoudakis, F. Christianos, A. Rahman, and S. V. Albrecht, “Dealing with Non-Stationarity in Multi-Agent Deep Reinforcement Learning,” *arXiv [cs.LG]*, 11-Jun-2019.

[10] S. Devlin, “Potential-Based Reward Shaping for Knowledge-Based, Multi-Agent Reinforcement Learning.pdf,” 2013.

[11] W. Daniel HILLIS, “Co-evolving parasites improve simulated evolution as an optimization procedure,” *Physica D*, vol. 42, no. 1–3, pp. 228–234, 1990.

[12] E. De Jong E Popovici A Bucci R Wiegand, “Coevolutionary Principles.pdf,” 2012.

[13] T. G. Tan, J. Teo, and H. K. Lau, “Competitive Coevolution with K-Random Opponents for Pareto Multiobjective Optimization,” *Third International Conference on Natural Computation (ICNC 2007)*. 2007.

[14] R. A. Watson and J. B. Pollack, “Coevolutionary Dynamics in a Minimal Substrate,” 2002.

[15] D. Cliff and G. F. Miller, “Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations,” *Advances in Artificial Life*. pp. 200–218, 1995.

1. https://github.com/TomClarkJorvik/Tom-Clark-CS-BEng-Project [↑](#footnote-ref-1)