# Finding a balance between reinforcement and evolution

Filippo Balzarini[a], Jason Kaxiras[a], Melvin Gode[a]

[a] *Department of Computer Science, Uppsala University, Uppsala, Sweden*

May, 2024

**Abstract**

This paper explores the comparative advantages and disadvantages of using a genetic algorithm versus a reinforcement learning approach for the pole balancing problem. Both methods were subjected to identical training and testing conditions to ensure a fair comparison. The results demonstrated that reinforcement learning, due to its simplicity of implementation and superior environmental comprehension, outperformed the genetic algorithm in this scenario. This finding reinforces the prevalent use of reinforcement learning in similar contexts and highlights its potential for applications in dynamic environments.

## 1. Introduction

In recent years, the fields of Reinforcement Learning (RL) and Genetic Algorithms (GA) have garnered significant attention for their ability to tackle complex optimization problems. Both techniques, despite their distinct methodologies, are really good in solving reinforcemnt learning problems. That is due to their main function [**taylor2006comparing**].

More specifically interesting is the comparison of the performance of Genetic Algorithms (GA) and Reinforcement Learning (RL) techniques in the context of a game environment. On one hand, in reinforcement learning the agent engages a dynamic and evolving environment by taking actions that affect it to accomplish a specific goal (solve an RL problem). On the other hand, we have evolutionary algorithms that employ evolutionary principles for automated and concurrent problem-solving by drawing inspiration from populations of evolving organisms. Despite their apparent dissimilarities, RL and GA both tackle the same fundamental issue: optimizing a function. This entails maximizing an agent's reward in RL and the fitness function in evolutionary algorithms, respectively, particularly in environments where the parameters may be unknown [**drugan2019reinforcement**].

One prevalent use of genetic algorithms often lies in optimizing multiparameter functions. Numerous problems can be framed as a quest for the optimal value, where this value represents a complex function of various input parameters. [**forrest1996genetic**]. More specifically *temporal difference methods* , a division of RL, learns a value function, which estimates the anticipated long-term reward associated with taking a specific action in a given state. Alternatively, genetic algorithms (GAs) offer another avenue for addressing RL problems by exploring the policy space to identify the one yielding maximum reward [**taylor2006comparing**].

This paper compares Reinforcement learning and Genetic Algorthms by having them balance a cartpole in 500 moves. More specifically it is a problem in nonlinear dynamics where an inverted pendulum is balancing in a cart. The aim or final goal of both RL and GA are to keep it the system balanced until they run out of moves. A graphic representation can of the environment can be seen in Figure 1. The environment will be described in more detail under the environment part.

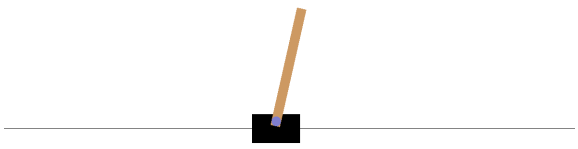

**Figure 1:** The cartpole in 2D graphics

## 2. Background

### *2.1. Reinforcement Learning*

Reinforcement learning is defined as the problem that an agent tries to solve by learning behavior through trial and error with its environment. In

other words, programming an agent through rewards and punishments rather than how to specifically solve the task itself [**kaelbling1996reinforcement**] as depicted in Figure 2.

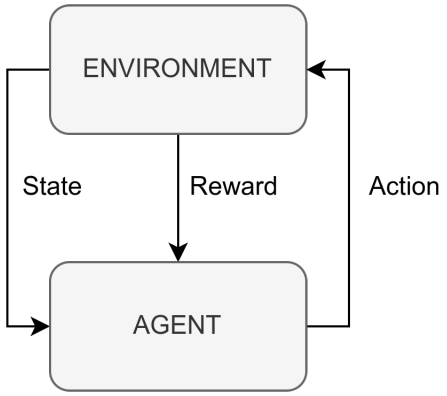


**Figure 2:** Graph representing reinforcement learning

The first concept crucial for reinforcement learning is the *reward function* which is objective feedback from the environment. It is usually scalar values that are associated with state-action pairs. High rewards are usually associated with state-action pairs which beneficial for the agent to be situated in whereas negative rewards would then be disadvantageous states or *hazardous* for the agent to be in. Essentially what is good and bad for the agent in the environment. The sole objective of the agent is then to maximize this reward [**sutton1999reinforcement**].

Naturally, we have to define *state* and *action*, which compared to the rest of the concepts have a very general definition. That being the latter is a descision of some sort and the former a factor that must be considered when taking an action.

#### *2.1.1. Temporal difference learning*

A central class of methods in reinforcement learning is *temporal difference learning* (TD). It refers to a class of methods in which the learning is based on the difference between temporally successive predictions. It aims to adjust the learner's current expectation for the present input pattern so that it more accuratly aligns with the subsequent prediction at the following time step. Unlike Monte Carlo methods and other methods in temporal learning updates its estimated value function at every step. [**tesauro1995temporal**]. TD methods learn a value function based on a state-action pair, which estimates the expected long-term reward if said state-action pair is taken.

In temporal learning there are several submethods or rather algorithms such as SARSA ,Q learning, TD-Lambda, and more [**eiben2007reinforcement**].

#### *2.1.2. Q learning*

Q learning is an algorithm where the environment can be constituted by a controlled Markov process where the agent is controlling it [**watkins1992q**]. The agent chooses an action and accordingly gets rewarded for it. It estimates the value of taking an action in a particular state based on immediate rewards and the current Q value. Q-learning uses the Markov chains to calculate the max reward that can be accumulated by the next state action and updates towards that as shown in the equation below.

$$Q(s,a) := Q(s,a) + \eta[r + \gamma \max Q(s',a') - Q(s,a)] \tag{1}$$

Equation 1 is the *value* or *update* equation which is responsible for mapping the different states based on their estimated long-term reward in Q learning. Here $Q(s,a)$ is the current state of the agent $r$ is the reward, $\eta$ is the learning rate and $Q(s',a')$ is the next state. An important variable here is $\gamma$ which represents the discount factor. This is used to limit the Markov chain to a limited finite number so they don't end up infinite. This controls how many steps into the future the agent will try to estimate.

### *2.2. Evolutionary computing*

Evolutionary computing comprises computational models based on the concept of evolution as seen in biology. It includes many different models but the most biologically accurate is Genetic Algorithms [**drugan2019reinforcement**] . Similarly to how organisms evolve by natural selection and sexual reproduction, programs can also simulate these processes and behave in a similar fashion to organisms in order to solve a specific problem. Natural selection is the process that determines which individuals get to survive by some test of fitness. After the best fits are selected the creation or reproduction of the next generation starts. Reproduction is then the method in which the mixing of genes in the remaining population happens and gets passed to the offspring [**holland1992genetic**].

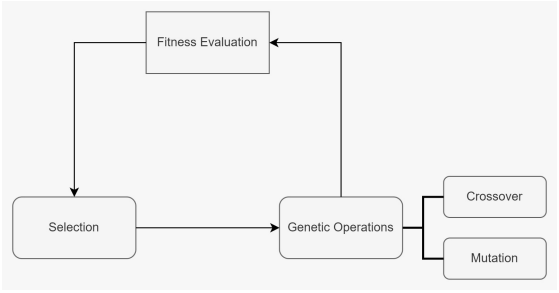


**Figure 3:** visual representation of genetic algorithms

By starting with a randomly created population, we have an initial population with variation amongst the individuals. The DNA which is essentially the code of the gene can be represented by a string of bits. These string bits can thought of as potential solutions to the problem. Due to the variation in the population, some individuals will be better *fit* which then will be selected to remain. In the final stage, the remaining individuals will mix their bit strings to produce individuals for the next generation. These steps will be continually done fore some number of generations [**forrest1996genetic**].

It is important however that we reduce the genetic drift and keep track of hte best solutions that have been produced by the previous generation. To do that we employ a method called Elitism. In elitism compared with traditional reproduction the most fitting individuals are copied to the next generation without any alteration. In that way the best solution of each generation is always preserved and adds selective pressure and improves convergence speed [**du2018elitism**].

### *2.3. Enviroment*

The environment that will be used to compare the two methods is called cart pole. It is based on the inverted pendulum, a classic control problem that received attention in the field of reinforcement learning in the eighties. The inverted pendulum balances on a cart that ca go either left or right on a horizontal line. The objective is to maintain the balance of the pole by moving the cart. However the movment should be such so the poll remainsbalanced[**moriarty1996efficient**].

The goal is to balance a pole on a cart that can move left or right.

The state space is formally described in Equation 2, where every vector $s \in \mathbb{S}$ $(p, v, \alpha, \omega)$, where $p$ consists of a cart position, $v$ cart velocity, $\alpha$ pole angle, $\omega$ and pole angular velocity.

$$\mathbb{S} = \left\{ s \in \mathbb{R}^4 \mid -4.8 \leq p \leq 4.8, -24^\circ \leq \theta \leq 24^\circ \right\} \tag{2}$$

The action space is discrete, with two possible actions: move left or move right.

The reward is 1 for every time step the pole is mantained balanced.

The goal is to balance the pole for as long as possible, with a limit of 500 actions.

The environment, called *CartPole-v1*, is implemented in Python using the Gymnasium library [**towers_gymnasium_2023**].

## 3. Method

The below described implementations have been trained ensuring that in every iteration the starting point is the same between two methods, in order to evaluate the two algorihtms in the exact same conditions.

### *3.1. Reinforcement Learning*

The reinforcement learning implementation is based on temporal differenc learning [**sutton1998temporal**], in particular Q-learning. The implementation takes inspiration from the work of *JackFurby* [**JackFurbyCartPole**].

The *Q-table* is represented by the discretization of the continuous 4-dimensional state vector in 20 even intervals for every dimension of the vector leading to 160000 possible pairs of $< state, action >$, considering the two possible actions, Figure 4 shows an example of the explained state-discretization.

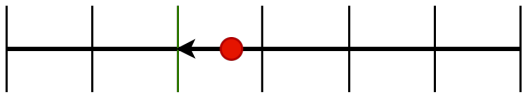

**Figure 4:** Representation of the state discretization technique, considering an element of the state, $s_i$, the red dot is the real value of $s_i$, this is discretized to the nearest leftwise discrete state.

Once an action is performed, the state selected is the first larger that the observed state.
The parameters used in the experiments are the following:

### *3.2. Genetic Algorithms*

#### *3.2.1. Genotype*

Since Genetic Algorithms can be very different depending on the genotype chosen to represent individuals, we have tried several different implementation of GA, varying the used genotype.

The first method used is a very naive implementation that can be applied to a very large variety of problems with GA : reprensenting individuals with the vector of all actions they will perform in order.

| | |
|---|---|
| **Learning rate** $\alpha$ | 0.1 |
| **Discount factor** $\gamma$ | 1 |
| **Number of episodes** $n_{ep}$ | 20000 |
| **Exploration rate** $\varepsilon$ | decaying |
| **Penalty factor** $PF$ | -375 |

**Table 1:** Parameters used in the RL implementation. The exploration rate $\varepsilon$ starts with $\varepsilon(0) = 1$ and decays by $\varepsilon(t) = \varepsilon(t-1) - \frac{1}{\frac{n_{ep}}{2}-1}$, every episode, stopping after $\frac{n_{ep}}{2}$ episodes. The penalty factor penalize the Q-value of the determinated state-action tuple of a value $PF$

| | |
|---|---|
| **Genotype** | Q-table |
| **Population Size** | 100 |
| **Generations** | 200 |
| **Selection** | Fitness |
| **Mutation Rate** | 0.005 |
| **Crossover** | one-point |
| **Elitism** | 2 |

**Table 2:** Parameters used in the GA implementation, the *Elitims* parameter describe how many individual from the last generation are saved for the successive one

Thus, *i-th* character of the genotype of an individual $j$ corresponds to the *i-th* action performed by the corresponding individual. In this apporach, mutation is performed by switching an action in the genotype from left to right or from right to left with a probability given by the *Mutation rate* for every action $i$ inside the genotype.
Since this particular genotype does not generalize well with the random initialization of the starting position of the pole. Due its intrinisc dependecy with the initial state, fixed starting conditions should be applied to obtain good results with this genotype, forcing the environment to always start at the same place, but this leads to a scarse ability of generalization, since the training is valid just for a determinated starting position of the pole.

All those considerations lead to the decision of evaluating other encodings for the final implementation.

The second encoding takes inspiration from Reinforcement Learning *Q-table*. In this approach, the focus is not to predict every action individually but instead use GA to assign values to state-actions pairs then select the action that referes to the observed state.
The discretization techninque mirrors that utilized in the reinforcement learning implementation and briefly outlined in Figure 4.
Here, mutation is performed by swapping the action of a given state with a probability determined by the *Mutation rate*.

#### *3.2.2. Parameters*

The GA parameters can be found in the following table :

The one-point crossover method has been adopted, where the state-action pair is divided precisely at its midpoint. Consequently, for the first offspring, the initial portion of the table inherits traits from the first parent, while the latter part derives from the second parent. Conversely, the second offspring exhibits the reverse pattern, inheriting the initial traits from the second parent and the latter traits from the first.

## 4. Results

### *4.1. Training comparison*

The training phase of a Reinforcement Learning agent and a Genetic Algorithm are fundamentally different. Therefore, a way to harmonize the training data of the two methods in order to compare them has to be found.
The first modification is to consider RL episodes the same as GA individuals and aggregate the RL performances to match the number of generations used for GA. For example considering a population size of $k$ for the Genetic Algorithm, the max and mean of every $k$ Reinforcement Learning episodes have been considered to compare them with each GA generation. Here, $k = 100$.
Second, the runtime per iteration might not be the same between the two methods. To account for this difference, the performance data was plotted not against the number of iterations but against the training time (in seconds). Of course both algorithms were ran on the same machine for a fair comparison.
Figure 5 and Figure 6 show both average and best results achieved over training time for our two methods.

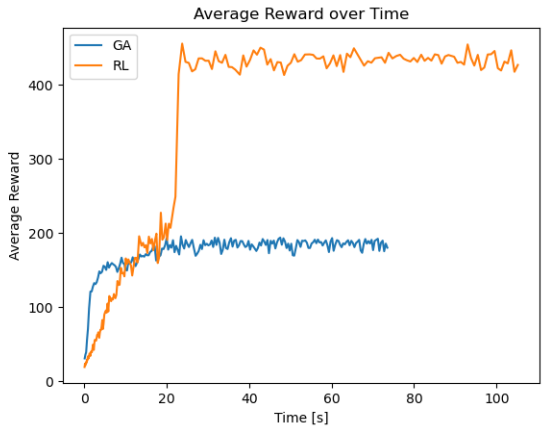


**Figure 5:** Evolution of mean score in time of both the two algorithms.

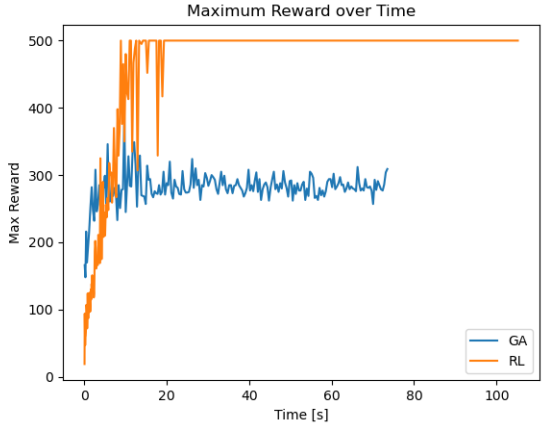


**Figure 6:** Evolution of max score in time of both the two algorithms.

As can be seen in Figure 5 and Figure 6, the first takeaway is that RL scores much better than GA. One interesting point though, is that GA performances seem to grow quicker than their RL counterpart at the beginning of training. This can be explained by the fact that GA performs parallel search and that the genotype can be modified much more in one crossover than RL's Q-table in an episode. These two factors create more variety in the achieved solutions and by retaining the best out of them, it is easy to see how we might arrive earlier to a viable solution.
However, the other side of this described coin is that, with less stability provided by GA, comes a harder time in making specific and iterative changes to a solution. Looking at figure 5, it is very interesting to note these very similar performances between the two algorithms at 10 to 20 seconds of training, before RL jumps much higher while GA keeps plateau-ing at the same level. An explanation to the phenomenon might be that RL iteratively tweeks the same solution to perform better which allows to fix encountered problems. Meanwhile, GA might provide too much change over each generation to slowly improve an existing solution further, explaining the flattening out of performance.

*4.1.1. Angle magnitude difference.*
Another difference between the two approach is the behavior of the agent during the game itself. This was only made visible once GIFs were generated to view sample games, as resumed in Figure 8 for the reinforcement learning agent and Figure 7 for the genetic algorithm last generation. It became apparent that the RL trained agent was taking way wider swings and generally playing more "on the edge" than the GA trained agent who is keeping the pole more straight and moving more cautiously. This initial obsevation was confirmed more quantitatively, by measuring the average absolute value of the angle of the pole during every episode of training. Indeed, it turns out that the average absolute value of the angle is consistently two orders of magnitude greater during RL training than during GA training. Since it doesn't penalize exploratory moves, Q-Learning specifically is known to be more prone to taking risks than other RL techniques. However, we tried using SARSA (another RL method without this feature) and still confirmed the same results.
This last remark might be even more surprising considering that we know the RL agent clearly outperforms the Genetic Algorithm, we might intuitively think that keeping the pole more straight would reflect better overall play.

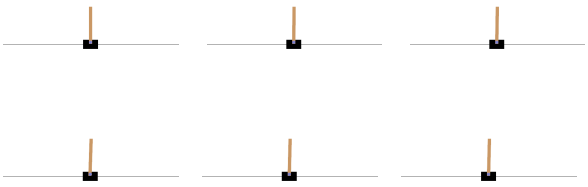

**Figure 7:** 6 early frames of a game played by the Genetic Algorithm last generation, as can be seen the agent is keeping the pole straight with very little movement.

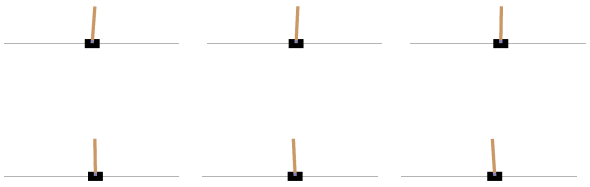

**Figure 8:** 6 early frames of a game played by the Reinforcement Learning agent, as can be seen the agent is taking wider swings and playing more "on the edge" than the GA trained agent.

*Since displaying one image every frame is very little change and hard to observe without showing too many of them at once, the images above are picked every 3 frames of a played game. They are Respectively frames number 6, 9, 12, 15, 18 and 21.*

### *4.2. final model comparison*

The second approach used to compare the obtained results is based on the final models' observations.

#### *4.2.1. State-Action table*

Figure 9 shows the difference between the two obtained state-action tables of the final reinforcement learning agent and one of the individuals from the last generation of the genetic algorithm.

Since the *Q-Table* does not provide an exact chosen action given a state, the compared state-action table of the reinforcement learning agent used in Figure 9 is obtained by selecting the action $a$ of the state $s$ based on $argmax_{a_i}(Q(s, a_i))$.

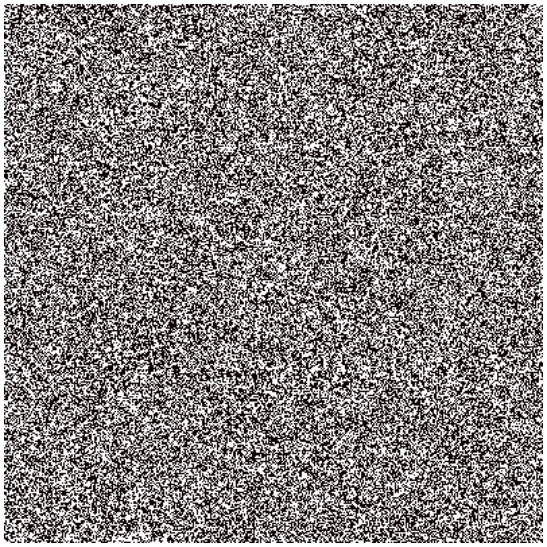

**Figure 9:** Difference between the two state-action tables obtained using GA and RL. The black pixels represent the states where the chosen action is the same, while white pixels represent a different action choice. The x-axis contains all the possible pairs of the first two elements of the state's 4D-vector, $(s_1, s_2)$, and the y-axis contains all the possible pairs of the last two elements, $(s_3, s_4)$.

Looking into Figure 9, it is possible to observe how the actions selected by the two algorithms differ in various states. However, some wide areas where the action taken by both algorithms is the same can be spotted, which could be referred to as sensitive states where considering a different choice could lead the pole to fall down.

#### *4.2.2. Testing performances*

Figure 10 shows the final testing results of the two methods. As already showed by the traning comparison in both figure 5 and figure 6, The reinforcement learning agent showed a better generalization ability and more understanding of the environment, leading to obtain better results.
The testing has been conducted as follow: the environment has been seeded with ten different seeds and for every execution the same environment has been provided to both the algorithm. The reinforcement learning agent has been tested using the obtained *q-table* from the training, the genetic algorithm population is the last generation of the training performed.
Every individual of the GA population have been tested for all the tests, and the best one has been considered for the comparison, during the testing of the population, no particular difference have been noticed between the reward obtained by the individuals.

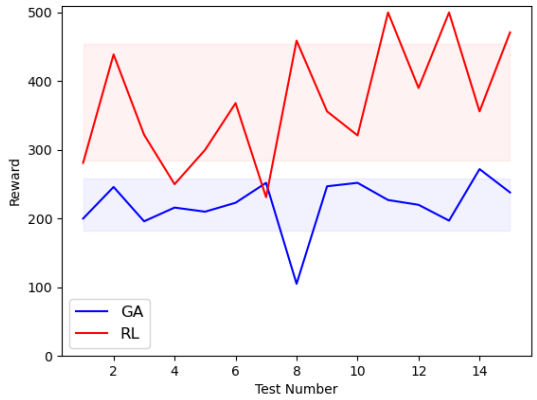


**Figure 10:** The plot shows the results obtained testing the two models on the same test set. The x-axis represents the number of the test set, the y-axis the number of steps the model was able to take before reaching the goal. The blue line represents the results obtained using the model trained with the GA, the red line the results obtained using the model trained with the RL.

Observing the results, some differences regarding the variance can be noticed: the GA population seem to obtain more consistent results in different tests, but due the fact that the testing is performed on the entire generation this could be related to the fact that every time the best individual is extracted.

## 5. Related Works

Similar comparisons between RL and GA have been made by [**drugan2019reinforcement**],[**taylor2006comparing**] and [**pollack1997coevolution**]. [**drugan2019reinforcement**] focuses more on a comprehensive overview of recent trends in the field rather than comparing subclasses of algorithms or particular aspects of RL and GA and states that both can be good at solving RL problems. [**pollack1997coevolution**] successfully trained an evolutionary algorithm to play backgammon, a typical RL type game. [**moriarty1996efficient**] presents a reinforcement learning technique SANE(Symbiotic, Adaptive Neuro-Evolution) which uses a genetic algorithm to evolve a population of neurons to perform a task. The evaluation which was done using the inverted pendulum problem in the form of a cart pole. Using the same problem it was compared to Q learning and was found to be 2 times faster. Several works focus on combining these two methods for machine learning by either using GA to train RL or vice versa such as [**eiben2007reinforcement**] where the authors try to use Reinforcement learning to tune the parameters of GA. Papers such as [**khadka2018evolutionary**] explore the opposite combination of training RL using GA . It is important to mention that the implementation of the reinforcement learning algorithm that is used is based on the work of *JackFurby* [**JackFurbyCartPole**].

In our investigation into various genotype options for addressing the pole-balancing problem, we consciously opted not to explore a solution based on NEAT[**stanley2002NEAT**] networks, build a neural network able to predict the correct action given the environment state as input.
Considering the simplicity of the chosen environment and given our primary objective of comparing genetic algorithm (GA) approaches with reinforcement learning (RL) agents, we excluded the possibility to employ a NEAT network for such environment.

Ensuring a fair and transparent comparison between GA and RL methods is the main objective. Introducing unnecessary complexity through a NEAT network could potentially obscure the true comparative performance of the two approaches. Thus, we chose simpler solutions, aligning more closely with typical RL agent implementations. This approach facilitates a clearer evaluation of the relative effectiveness and efficiency of GA and RL algorithms in the pole-balancing task.

## 6. Discussion

Originally, when only trying the action-by-action approach in GA, the dependency of the genotype on the initial state led to poor performances for GA implementation and no real generalization capacities. However, it was very interesting to see that the Genetic Algorithm performed much better when combining it with features inspired from Reinforcement Learning - namely the *Q-table*.
One thing that we also realized while working on this project is also the fact that the parameter search space is much larger in GA than for its RL counterpart. While RL appears to be doing approximately the same thing at a difference pace depending on its parameters, it can feel like GA behaves in very different ways depending on the chosen combination of parameters. While probably not very determining in the final results, this could also partly explain why we did not manage to achieve the same performances with GA as witn RL.

### *6.1. Future work*

A few ideas that we had which could probably improve GA's performance would be to try to target

mutations more efficiently. First, we could increase the probability of mutation for low scoring individuals (note that we would then have to play the game twice per generation to measure fitness after crossover). This is not a new idea in Genetic Algorithms and is a very reasonable thing to do.
Second, thinking about the fact that RL is better at targeting the areas it needs to improve on, we thought about increasing the probability of mutation, specifically for states close to where individuals often lose. This could help "focusing" on what is wrong with our genotype and performing smarter mutations.
However it is important to keep in mind that, for Genetic Algorithms, the crossover is the most important search operator, while mutation is only supposed to bring some diversity and little nudges in the search. So we should not expect too much out of improving the mutation operation as it is not the main point of the algorithm.

It is also good to keep in mind that the obtained results could be possible thanks to *state discretization*, which allowed us to use the aforementioned tabular approach. Without the state discretization approach, different and more elaborate directions should have been considered.

## 7. Conclusion

The experiments concluded that Genetic Algorithms can be an interesting pathway to parametrize features inspired from Reinforcement Learning (here the *Q-table*). However RL still vastly outperforms GA on the studied task, supposedly due to the precision required to improve pole balancing and a very large hyper-parameters search space. GA may thus be better suited (and possibly outperform RL) on tasks with a larger search space or requiring less precision, where its parallel search could allow it to converge to a viable solution faster while not being impaired by its "blind" update steps.