Comparing the Performance of Genetic algorithm with Reinforcement learning for simple game learning problem

Group Members: Praveen Tangarajan – PXT190003 & Shariq Ali – SXA190016

1) Introduction:

Reinforcement learning is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize some notion of cumulative reward. It basically attempts to maximise the expected sum of rewards as per a pre-defined reward structure obtained by the agent. It does so by learning a value function which is updated using information obtained from the agent's interactions with its environment. The interactions are dictated by a balance of actions for exploration and exploitation. Exploitation ensures that the agent makes use of (exploits) the already acquired knowledge (value function), while exploration makes sure the agent remains 'playful', which might allow it to improve its knowledge.

On the other hand Evolutionary Computing algorithms like Genetic algorithm starts with a population of randomly generated individuals/solutions, and uses the principle of natural selection to discover useful sets of solutions. The selection is usually fitness, with fitter individuals being allowed a higher probability of being selected into the subsequent generation. In order to search the solution-space, a proportion of individuals are subjected to mutation and crossover operations. The hope is that as the generation progress, fitter and fitter individuals get selected into the subsequent generations, which will ultimately deliver optimal or close to optimal solutions.

These are the fundamental difference between these two learning algorithms:

- The fundamental operating of the two approaches for both the algorithms is very different. Reinforcement learning uses a relatively well understood and mathematically grounded framework of Markov decision processes, whereas most evolutionary computing algorithms like genetic algorithms etc are largely based on heuristics.
- The value function update in reinforcement learning is a gradient-based update, whereas genetic algorithm generally doesn't use such gradients.
- Reinforcement learning uses the concept of one agent, and the agent learns by interacting with the environment in different ways. In evolutionary algorithms, they usually start with many "agents" and only the "strong ones survive"
- Reinforcement learning benefits from features such as temporal credit assignment, which can speed up the convergence and other mechanisms such as experience replay, wherein recent agent-environment interaction information is stored and recalled from time to time in order to accelerate learning whereas no such strategies exists in Genetic algorithms.
- Reinforcement learning agent learns both positive and negative actions, but evolutionary algorithms only learns the optimal, and the negative or suboptimal solution information are discarded and lost.

Despite these differences, one may argue that there are certain similarities between the two approaches like they are both nature-inspired, and they both attempt to find goodness of solutions as per some pre-defined notion of 'goodness' i.e. Reinforcement Learning and Evolutionary Computing are two different optimisation techniques that address the same problem: the optimisation of a function, the maximisation of an agent's reward in reinforcement Learning and the fitness function in evolutionary computing , respectively, in potentially unknown environments. In terms of utility, recent studies (Evolution Strategies as a Scalable Alternative to

Reinforcement Learning) have suggested that GA approaches could serve as a reasonable alternative to RL ones (at least for some problems). However, current research directions indicate that these two algorithms are related in a more subtle, but important sense - in the sense that it is a combination of some versions of these algorithms (along with other supervised and unsupervised algorithms) which could potentially lead to an artificial general intelligence (AGI).

This project aims to compare the performance of these two different strategies on a specific game learning task. The learning agent basically tries to learn and find solutions to the problem.

2) Problem definition:

In this project we compare the performance of Evolutionary computing algorithm like genetic algorithm with reinforcement learning like Q-learning for a simple game learning problem, the game learning problem setup is given a graph and a start and end location the goal of the learning algorithm/agent is to find the best possible path from the source to destination. The testing has been done on various graph setups of different sizes, sparseness and even by blocking few nodes/path in the graph. Only a specific version of genetic algorithm and reinforcement learning algorithm is applied to this specific problem and all conclusions and results derived are restricted within this scope and might or might not scale well for other problems.

3) Problem Setup:

Consider the game setup similar to a simple maze problem. Basically maze is a path or collection of paths, typically from an entrance to a goal. The word is used to refer both to branching tour puzzles through which the solver must find a route, and to simpler non-branching patterns that lead unambiguously through a convoluted layout to a goal. The input for this problem is defined as an adjacency matrix which represents connectivity from one location to another. Given a start location, the goal of the algorithm is to find the best path (shortest path in terms of nodes) from the start to the given end location. We also block certain intermediate nodes in the graph which indicate a blockage similar to the maze problem.

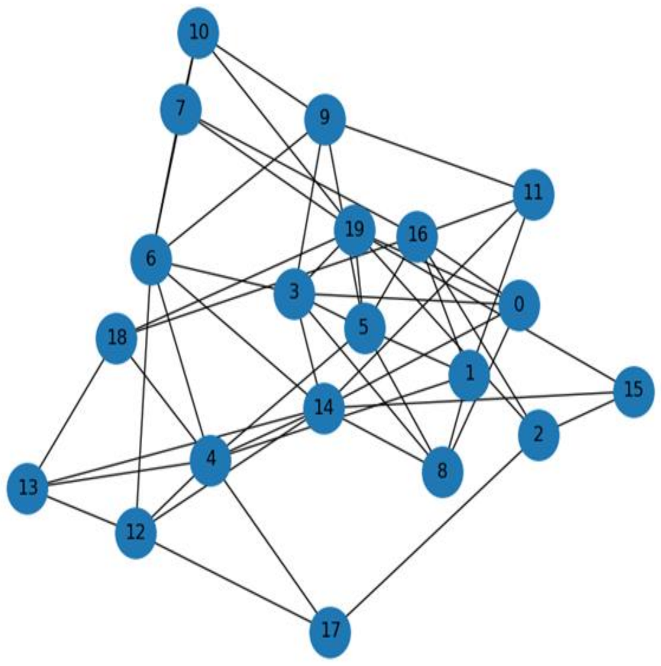


Figure 1 - Unblocked

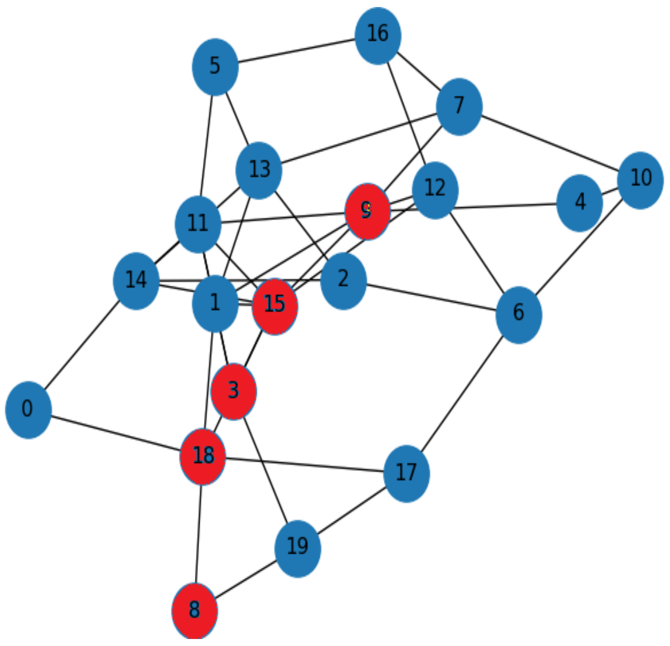


Figure 2- Blocked

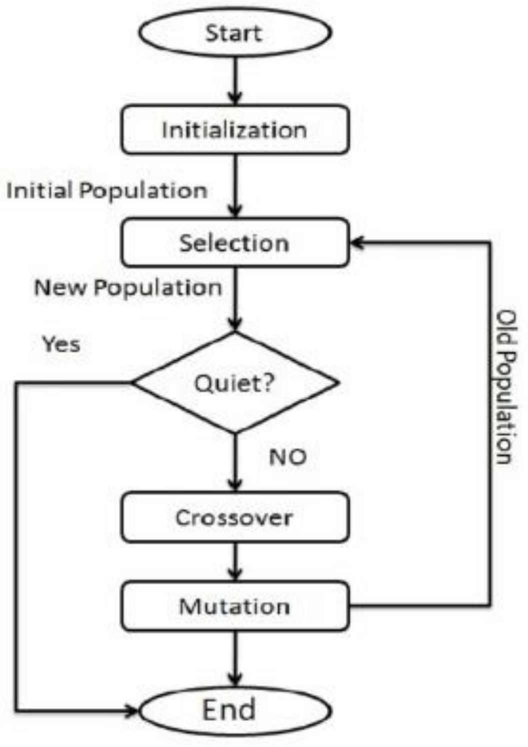
Consider the graph the above, the task of the learning algorithm in each of the case is to find shortest path from the start node 0 to the end node 19. The first figure is an unblocked case whereas in second case the nodes indicated by red colour is blocked which means the algorithm cannot pass through these marked nodes to reach the destination.

4) Algorithms:

Two different algorithms one belonging to family of evolutionary strategies and one belonging to reinforcement learning have been used. In this research, we use specific version of genetic algorithm, a type of evolutionary computing algorithm and q-learning algorithm which is a type of reinforcement learning to solve the given game learning problem.

4.1) Genetic algorithm:

The algorithm basically starts with a population of randomly generated individuals/solutions, and uses the principle of natural selection to discover useful sets of solutions. The selection is usually fitness, with fitter individuals being allowed a higher probability of being selected into the subsequent generation. In order to search the solution-space, a proportion of individuals are subjected to mutation and crossover operations.



Input setup: For the given problem we set up the genetic algorithm as a minimization problem, in the given adjacency matrix, a path from one node to other is indicated by 1 or 0 otherwise. We block the nodes by setting up higher weights for each of the edges in the node.

For the corresponding problem setup we elaborate how each of these steps was implemented:

Initial Population: For the initial set of population, given a start node and end node, k population of solutions are generated. The populations are generated by using a random walk from start node until it reaches the end node. Consider the figure 1, an example of random solution from 0th node to 19th node would be [0,8,14,6,3,19]

Fitness Function: The fitness function in this case calculate the distance of the route which is then used to minimize the distance from each node to each other node. In this particular case we assign constant distance as 1 for connections. This should also work if we give real valued weights or distances in numbers. For example the fitness value for the route [0,8,14,6,3,19] in figure 1 would be 5.

Cross over operator: In this process we take two solutions and mix them to find another solution. There are different cross-over operators available in literature like single point, multi-point, order based etc. For the problem we choose to use single point cross-over technique.

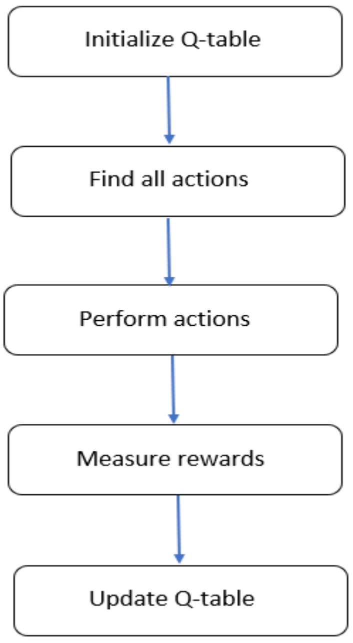
For example consider two different route solutions:
Solution1: [0,7,2,18,3,7,20]
Solution2: [0,6,2,18,3,2,18,3,7,20]
Let the cross-over point be 3.(typically we choose it randomly)
Resultant new solution:
Solution1: [0,7,2,18,3,2,18,3,7,20]
Solution2: [0,6,2,18,3,7,20]

Mutation operator: We introduce random noise within the solution by changing the values based on certain probability. Typically the mutation probability is kept low.

Selection operator: We have used random selection operator based on a probability values, there are other different types of selection operators like tournament, roulette etc which can be tried extending the work.

4.2) Q learning algorithm:

The Q learning algorithm is the simplest algorithm in reinforcement learning paradigm. It basically maintains a lookup table, called a Q-table, which keeps track of all the states and all possible actions from those states with the associated rewards. During training time we just chose a random node, find all possible actions from that node, calculate the reward associated for those actions and update the Q-table with the calculated reward from that state for each action. This forms the fundamental step of Q learning and is repeated until convergence. At convergence we have the path of maximum rewards from each node to the goal. During test time, the agent simply follows the path of maximum rewards.



Input setup: For the given problem we set up the reinforcement algorithm as a maximization problem, in the given adjacency matrix, a path from one node to the other is indicated by 0 or greater whereas the absence of a path is represented by negative weights. We block the nodes by setting up higher negative weights for each of the edges in the node.

Initialization: We will first make a N x N matrix, where N is the number of states, initialized with zeros. Each action corresponds to an edge from one node to the other. So for each node there will be N possible actions.

Find all actions : Then find all the valid actions from the current state based on the adjacency matrix.

Perform actions : Perform each action.
Measure Rewards: Calculate the rewards for performing that particular action from the current state based on the below formula:

*Q_table[current_state,action] = Adjacency_Matrix[current_state,action] + gamma * max_value*

where,
Q_table = an N x N table which contains max reward lookup value from each state
current_state = current state of the agent or node of the network
action = one of the valid actions at the current state
Adjacency_Matrix = contains information about the edges between each pair of nodes
gamma = probability of choosing the action with max_value at each step
max_value = maximum reward value out of all the actions at the current state

Update Q-table : Update the Q-table with the calculated maximum reward value from each state.

5)Experimental Evaluation:

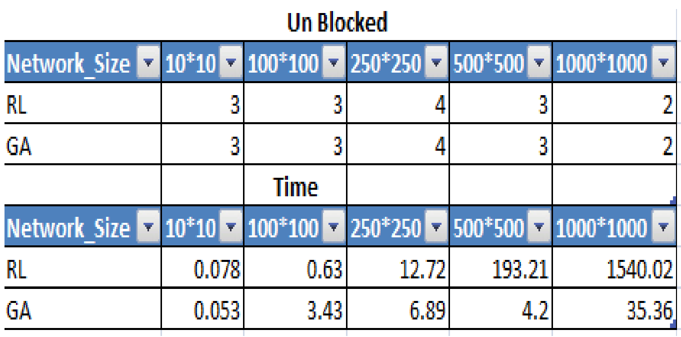
5.1) Methodology

The experiments are performed on network topologies of various sizes, i.e. number of nodes in the network ranging from 10, 100, 250, 500 & 1000. For each of the algorithms and topology combination results were ran multiple times typically 10 times,(as these algorithms include randomization) and the minimum path length from start node to the end node that was obtained maximum number of times was recorded and taken as the best path. Similar experiments were performed by blocking the multiple nodes in the path and results were noted.

These experiments were performed in a Hexa-core i7 processor with 2.60 GHz speed and 16 GB ram. The time for convergence of each of these were noted and compared. The following metrics were compared to arrive at the final results on the performance of the algorithm.

5.2) Results:

We compare the results of path length and running time for each of the network-algorithm combination for blocked and unblocked criteria.

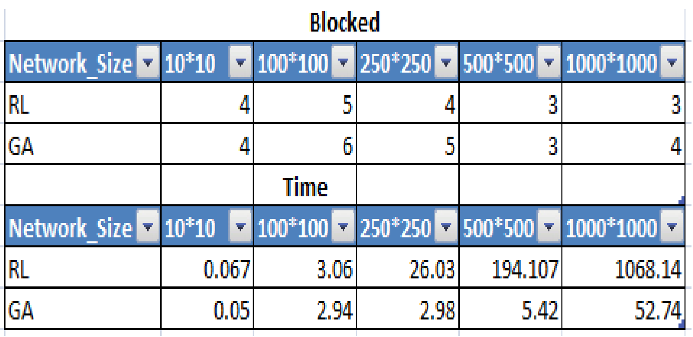


Un Blocked

Network_Size	10*10	100*100	250*250	500*500	1000*1000
RL	3	3	4	3	2
GA	3	3	4	3	2

Time

Network_Size	10*10	100*100	250*250	500*500	1000*1000
RL	0.078	0.63	12.72	193.21	1540.02
GA	0.053	3.43	6.89	4.2	35.36



Blocked

Network_Size	10*10	100*100	250*250	500*500	1000*1000
RL	4	5	4	3	3
GA	4	6	5	3	4

Time

Network_Size	10*10	100*100	250*250	500*500	1000*1000
RL	0.067	3.06	26.03	194.107	1068.14
GA	0.05	2.94	2.98	5.42	52.74

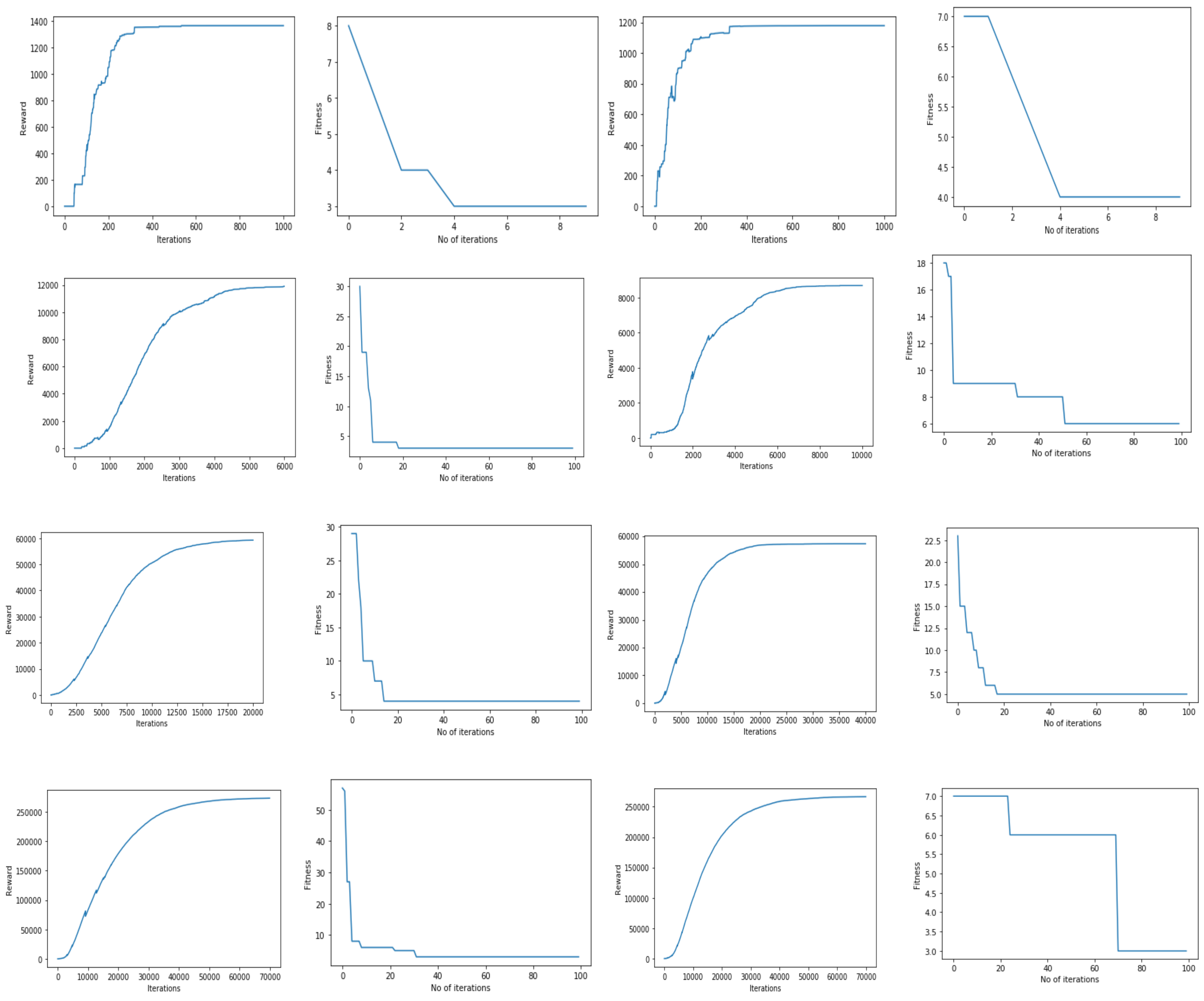


Image: Following are the convergence graphs

<u>Note:</u> Please find extensive experimental results in Results.docx

Conclusion: Based on the following results and we can conclude that both of the above mentioned algorithms were able to get more or less same results and they were able to converge properly. Genetic algorithm was faster than Q-learning algorithm, but the latter was more accurate for this particular problem type.

Related works:

There has not been much of related works in comparing these two algorithms for game learning problems. However, there are different approaches which are currently being tried like using combination of these algorithms like (neuro-genetic evolution strategy) or using evolutionary strategies to learn the weights of neural networks etc.

Future work:

The research presented in this project just compares a specific version of evolutionary algorithm to a specific version of reinforcement learning algorithm to a specific game learning problem. It seems an open-ended area of research. Future works can basically try different evolutionary algorithm or probably use the same algorithm mentioned in this research with different selection, crossover or mutation operator combination and further compare it with other different versions of reinforcement learning algorithms like SARSA, Q-learning, TD and further comparing it with different game learning tasks to measure the relative performance of these approaches.

Bibliography/Citations:

The following are the related papers that were referred:

1) Evolution Strategies as a Scalable Alternative to Reinforcement learning https://pdfs.semanticscholar.org/70f0/ea8459bc0229ed613aee43d1f08f5f5644aa.pdf
2) Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning https://arxiv.org/abs/1712.06567
3) How Learning Can Guide Evolution Geoffrey E. Hinton Steven J. Nowlan https://pdfs.semanticscholar.org/5d6c/84e7cd46d0a520ad6784a0f7f6825ef83685.pdf
4) https://tonytruong.net/solving-a-2d-maze-game-using-a-genetic-algorithm-and-a-search-part-2/
5) https://www.geeksforgeeks.org/ml-reinforcement-learning-algorithm-python-implementation-using-q-learning/
6) https://www.geeksforgeeks.org/deep-q-learning/
7) Evolutionary Algorithms for Reinforcement Learning https://arxiv.org/pdf/1106.0221.pdf
8) A Comparison of Genetic Algorithms and Reinforcement Learning for Optimising Sustainable Forest Management https://pdfs.semanticscholar.org/1c31/977ab609269aff0d58b64c2e28d4ff025ca8.pdf