Improved Genetic Algorithms For Reinforcement Learning

Ravi Haksar*
rhaksar@stanford.edu

Prafull Sharma*
prafull7@stanford.edu

Abstract

In this paper, we apply a variation of genetic algorithms, called grammatical evolution, to a single-agent and a multi-agent benchmark problem. RockSample problem is used as the single-agent problem, and baseline scores are generated from common POMDP solvers as well as two heuristic expert-provided policies. For the multi-agent problem, a novel problem is proposed, called BorderCross, which models a multi-robot coordination problem. Results for both problems are presented, which suggest that augmenting grammatical evolution with other methods would be able to address the shortcomings of genetic algorithms.

1. Introduction

Reinforcement learning is an active field of research in the domain of artificial intelligence. One objective in this field is to find a policy (a mapping from states to actions) for agent(s) that have partially observable information in order to maximize a reward function. In general, policies can be found by using two methods: by searching the space of value functions or by searching a policy space [1]. Problems involving partially observable environments generally have a very large policy space, which makes it a challenging task to find good solutions. More recently, genetic algorithms have been shown to be highly effective in searching a large spaces to maximize or minimize a function. These methods are inspired by biological processes, such as Darwinian or Lamarckian evolution, in which competition between individuals in populations drives the evolution of different behaviors. In the reinforcement learning context, a fitting function is used to evaluate the fitness of individuals that are evolved through random processes.The best members of the population move to the next generation and other members undergo modifications, such as crossover and mutation to find children that will form the next generation. While genetic algorithms can handle large search spaces, they are prone to stagnating in local minima for long durations, which makes a policy search inefficient. A recent variation on genetic programming called grammatical evolution has been developed to simplify and extend the development of policies to more general data structures. While genetic programming relies on a tree-structure which can easily be recursively evaluated, grammatical evolution applies genetic operators to an integer string, which is mapped to a valid program code using grammars. This method greatly increases the versatility of genetic evolution, as it can be used with different structures and program languages.

In this paper, we apply grammatical evolution to a single-agent benchmark problem for reinforcement learning, the RockSample problem. We compare the performance of grammatical evolution and other methods like random rollouts, QMDP and SARSOP. We also propose an improvement to the performance of genetic algorithms by finding a suboptimal parametrized policy by evolving a policy and then using gradient ascent on the parameters to improve the policy performance. We also propose a new multi-agent benchmark problem based on coordination, called BorderCross, to determine the effectiveness of grammatical evolution on higher-dimensional problems. We discuss several methods to find centralized policies for the BorderCross problem.

2. Previous Work

There has been previous work on finding policies for POMDP problems by performing a policy search.

Hansen presents an approach to solve POMDP by representing a policy as a finite-state controller and improving the policy by searching the policy space [2]. This work discussed policy iteration and a heuristic search algorithm that focuses on regions of the search space that are reachable or likely to be reached from a given start state. In a paper by Moriarty et al., the authors discuss the application of evolutionary algorithms to reinforcement learning problems. The authors discuss the strengths of evolutionary algorithms for solving reinforcement learning like scalability to large state spaces, dealing with incomplete state information, and non-stationary environments [3].

Shimon discusses neuroevolution techniques and topology and weight-evolving artificial neural networks (TWEANNs) to solve for policies for reinforcement learning problems [4]. In a recent work by Beigi et al., the authors propose a new algorithm, Evolutionary Q-Learning (EQL) which increases the efficiency of the traditional Q-learning algorithm [5] by only updating the Q-function with the best evolved individual.

3. Problem Statement

3.1. RockSample Problem 7x8

RockSample problem is a classic benchmark problem for reinforcement learning algorithms. There are many variations of the RockSample problem and for this work we used the original formulation with a 7x8 grid shown in Figure 1. The objective of the agent is to maximize reward in this partially observable environment.

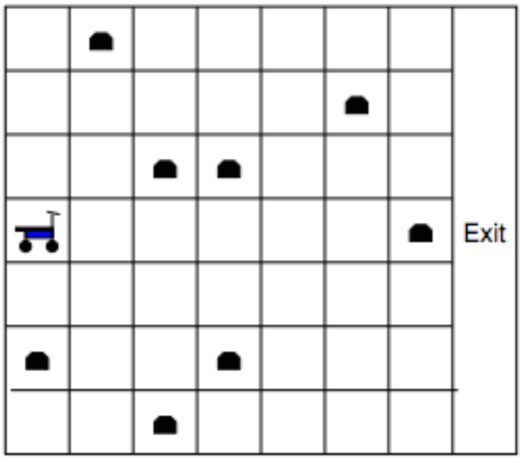

Figure 1: RockSample 7x8

State
In RockSample 7x8, the environment consists of a grid world of 7x8 cells and the agent begins at the center in the leftmost column, at position (1,4). There are 8 rocks in the environment which have fixed positions but are randomly assigned values of "Good" or "Bad." The rightmost column is the exit column which ends the problem as shown in Figure 1.

Action
The agent can move up, down, left and right which are equivalent to North, South, West, and East. It can also check any rock to get a noisy observation of the rock value. The objective of the agent is to sample rocks to obtain reward. Rocks that are "Good" turn "Bad" after being sampled. Sampling at grid location without a rock does not affect the environment.

Observation
The agent has a long-range sensor which returns a noisy observation of the value of the rock. The efficiency of the long-range sensor is defined as

$$\eta = 2^{-d/d_0}$$

where d is the distance to the rock and d_0 is a tunable constant called the half-efficiency distance. For this work, d_0 is set to 20 for the purpose of this paper. The observation result is either "Good" or "Bad".

Reward
The agent receives a +10 reward for sampling a "Good" rock and -10 reward for sampling a "Bad" rock. If the agent exits the environment by moving to the rightmost column, the agent receives a +10 reward. Illegal moves like moving outside the grid is penalized by a -100 reward.

3.2. BorderCross 7x8

To provide a baseline for a multi-agent problem, a novel test domain is presented. BorderCross, shown in Figure 2, requires two robotic agents to cooperate in order to cross the "wall" (denoted by 'X' in the figure) to reach the exit.

State
The gridworld is a fixed size with a 7x6 grid where the

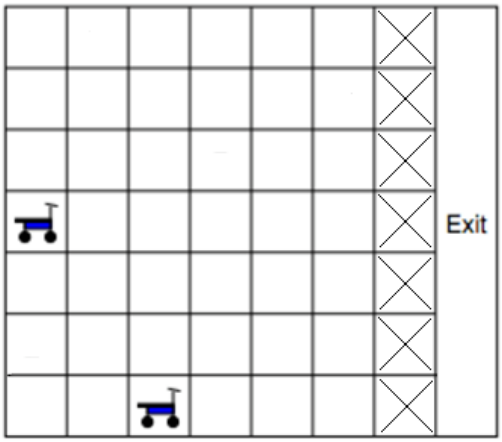

Figure 2: BorderCross 7x8

robots can move, with a wall that separates the agents from the exit, the rightmost column. The agents start at (4,1) and (7,3).

Action
Similar to RockSample, each robot can move in the four cardinal directions. A cooperate action is also available, which ends the game if the robots occupy the same cell and are adjacent to the wall.

Observation
A robot can calculate the Euclidean distance between it and the other robot, but this measurement is noisier the farther apart the robots are. The efficiency of the sensor is defined as

$$\eta = 2^{-d/d_0}$$

where d is the distance between robots and d_0 is a tunable constant (set to 20, as in RockSample). As the sensor becomes less efficient, the calculated distance becomes noisier: a small positive offset is increased and added to the distance calculation.

Reward
The robots both receive a reward of 250 for reaching the exist. Illegal moves, which are movements that lead to a robot leaving the grid, are penalized by a -100 reward.

4. Approach

4.1. RockSample Problem 7x8

The first part of this work is applying vanilla grammatical evolution to a single-agent and a multi-agent problem in order to create baselines to compare to other methods. Algorithm 1 shows the grammars used to solve RockSample, described in Backus-Naur Form (BNF) [7]. For grammatical evolution, it is fairly easy to provide expert or prior information.

Algorithm 1 BNF form for the RockSample Problem

procedure ROCKSAMPLE PROBLEM
N = code,line,expr,sub-expr,if-statement,op
T=left(), right(), up(), down(), sample(), check(), < rock > , < decimal > ,if
S = < expr >
P :
< code > ::= < line > | < code >
< line > ::= < expr >
< expr > ::= < if-statement > | < op >
< sub-expr > ::= < op >
< if-statement > ::= if(< condition >) sub-expr
< condition > ::= < rock > == "good" | belief(< rock >) > < decimal > l
< rock > ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8
< number > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
< decimal > ::= 0 + "." + < number > + < number >
< op > ::= left() | right() | up() | down() | sample() | check(< rock >)
end procedure

Several choices have been made to reduce the complexity of the full policy-space search. For one, there are no loops allowed, and the expressions evolved for if-statements cannot contain additional if-statements. Evolving nested statements, especially "for" or "while" loops, leads to a much larger policy space of which most of the policies are not intuitive results. Therefore, the choice was made to eliminate these. In addition, nesting if-statements often lead to logic chains that would never be satisfied or contradicted each other, leading to additional inefficiency in the search.

Two additional changes were made to the fitness evaluation to focus the search algorithm on more in-

tuitive solutions. First, the max number of steps possible in the simulation was limited to 50; any action in the simulation was considered a step (e.g. move, sample, check). The was done to introduce the idea of discounted rewards without keeping track of when the agent actually gathers reward from sampling. Second, a linearly increasing penalty was added for evolved solutions that exceeded 50 lines of code. This penalty discourages longer evolved codes which are more likely to be inefficient solutions.

In addition to searching the reduced policy space, two additional heuristic policies were created, which were parameterized by a single variable. Parameterized policies are amenable to policy gradient methods, which is a major focus of the proposed work in this paper. Therefore, a single-variable parameterized policy makes it simple to apply and analyze both grammatical evolution and gradient ascent, as well as the combination of both.

Algorithm 2 Accuracy-based policy

```
1:  procedure ACCURACY-BASED POLICY
2:      while not terminated, steps left do
3:          find nearest rock
4:          if valid rock then: get rock
5:          else: move east
6:          end if
7:      end while
8:  end procedure
9:
10: procedure FIND NEAREST ROCK
11:     for all uncollected rocks do
12:         if accuracy > λ and rock is "Good" then
13:             get distance
14:         end if
15:     end for
16:     return closest rock
17: end procedure
```

A description of the two parameterized policies are provided in Algorithms 2 and 3. Both algorithms look at the set of uncollected rocks and try to determine if the nearest rock should be collected. For the accuracy-based policy (Algorithm 2), the robot will collect the nearest rock if the calculated accuracy is above a threshold, which must be provided to the algorithm. The robot will also check if the rock is "Good" before attempting to collect it. If all rocks are collected, or if the rocks are sampled as "Bad", the robot will move to the exit.

Algorithm 3 Belief-based policy

```
1:  procedure BELIEF-BASED POLICY
2:      while not terminated, steps left do
3:          find nearest rock
4:          if valid rock then: get rock
5:          else: move east
6:          end if
7:      end while
8:  end procedure
9:
10: procedure FIND NEAREST ROCK
11:     for all uncollected rocks do
12:         if belief > γ then
13:             get distance
14:         end if
15:     end for
16:     return closest rock
17: end procedure
```

A similar method is used for the belief-based policy (Algorithm 3), but instead a belief is maintained that describes the probability that the robot believes a given rock is "Good". The belief update equations are described in Equations 1 and 2. A belief update is performed every time a rock is checked or sampled. Sampling a rock is deterministic, so the belief is changed to zero when a rock is sampled: if the rock was good, it is changed to bad, and the belief becomes zero; if it was bad then the belief is changed to zero as well.

$$b_{t+1} = \frac{\lambda b_t}{(1-\lambda)(1-b_t) + \lambda b_t} \tag{1}$$

$$b_{t+1} = \frac{(1-\lambda) b_t}{(1-\lambda) b_t + \lambda(1-b_t)} \tag{2}$$

where

$$\lambda = 0.5 + 0.5\eta \tag{3}$$

To verify the results generated for the parameterized policies by grammatical evolution, gradient ascent was used. Equations 4 and 5 describe the momentum-based gradient ascent, which is used to move past local maxima when searching for the maximum of a function. In practice, it was found that using a momentum-

update rate that decays with time lead to the algorithm converging. Gradient and momentum clipping were also used to avoid unstable oscillations that lead to poor results.

$$v = \gamma v + \alpha \nabla_\theta J(\theta) \tag{4}$$

$$\theta = \theta + v \tag{5}$$

To estimate the gradient of the parameterized policies, a simple first order forward difference approximation is made, which is shown in Equation 7. Using this approximation, the fitness function must be run twice, once with the original parameter, and once with the original parameter increased by a small positive constant.

$$\nabla_\theta f(\theta) = \frac{f(\theta + \epsilon) - f(\theta)}{\epsilon} \tag{6}$$

4.2. MULTI-AGENT PROBLEM

Algorithm 4 shows the Backus-Naur form for the BorderCross problem. Some simplifications are made to reduce the space of possible policies. In the form presented, the solutions generated are centralized, with one policy controlling both robots.

Algorithm 4 BNF form for the BorderCross Problem

1: **procedure** BORDERCROSS PROBLEM
2: N = code, line, expr, sub-expr, if-statement, op
3: T=left(), right(), up(), down(), sample(), distance(), < robot > , < decimal > ,if
4: S = < expr >
5: P :
6: < code > ::= < line > | < code > < line >
7: < line > ::= < expr >
8: < expr > ::= < if-statement > | < op >
9: < sub-expr > ::= < op >
10: < if-statement > ::= if(< condition >) sub-expr
11: < condition > ::= distance() < < number >
12: < robot > ::= 1 | 2
13: < decimal > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
14: < op > ::= left(< robot >) | right(< robot >) | up(< robot >) | down(< robot >) | cooperate()
15: **end procedure**

There is a significant difference between this problem and RockSample that inherently makes the problem more difficult. Due to the wall, it is very unlikely that the robots will take the right sequence of random actions to reach the sole source of positive reward which is reaching the exit. As a potential solution, reward shaping can motivate evolved solutions to perform the necessary intermediate steps to reach the exit. Below is the shaping model, which will added to the policy space search to create another method.

Reward Shaping
The robot receives a +1 reward for moving in the same cell as the other robot and +1 for moving to be adjacent to the wall.

Table 1: Results from Silver and Veness [6]

Rocksample	**(7, 8)**
States |S|	12,544
Rollout	9.46
SARSOP	21.39

5. Results

We applied random rollouts to the RockSample problem and got an average reward of around 12.8% of the maximum reward possible. Analyzing the runs, we discovered that the agent usually keeps moving East and then exits to receive a reward of +10 as well as occasionally sampling one "Good" rock. RockSample was also written to leverage the POMDPs.jl library to get another benchmark result to be compared with the alpha vectors for the QMDP policy and the the SARSOP policy. Belief updating was inefficient due to high dimensionality of the state space which made simulation of the policy difficult with the given computational resources. Silver and Veness discuss how SARSOP usually does almost twice as good as random rollouts as shown in the Table 1 [6]. Using that result it is possible to conclude that SARSOP would get at most 30% of the maximum reward for the RockSample problem. Since QMDP does not consider observations in its execution it is less efficient than SARSOP.

The simulation parameters used for the grammatical evolution algorithm are listed in Table 2. Figure 3 shows the results of running the full policy search for 500 generations, while the parameterized policies were run for 399 generations. While the full search makes steady improvements, the parameterized poli-

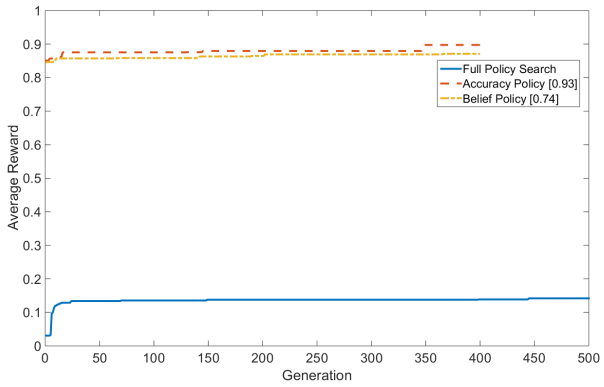

Figure 3: Results of the Grammatical Evolution Methods

Table 2: Simulation Parameters for Grammatical Evolution

Parameter	Value
Individuals in population	250
Episodes	100
Generations	500 / 399
Max steps	50
Penalty for long codes	max([0,length - 50])*0.1

cies quickly find a threshold (for Algorithm 2: 0.93, for Algorithm 3: 0.74) to obtain most of the reward. Note that although the parameterized policies show some improvement near the end, this is due to stochastic variation; the policies should be averaged over more episodes to obtain a more stable result.

```
if high_belief(rs,1,0.85)
  move(rs,"East")
end
sample(rs)
if is_good_rock(rs,1)
  move(rs,"South")
  move(rs,"North")
  move(rs,"South")
end
sample(rs)
```

Figure 4: Best generated policy by the full policy search

Figure 4 shows the best generated policy after 500 generations. The policy is not very efficient and relies on sampling the rock directly south of the rover when it is "Good".

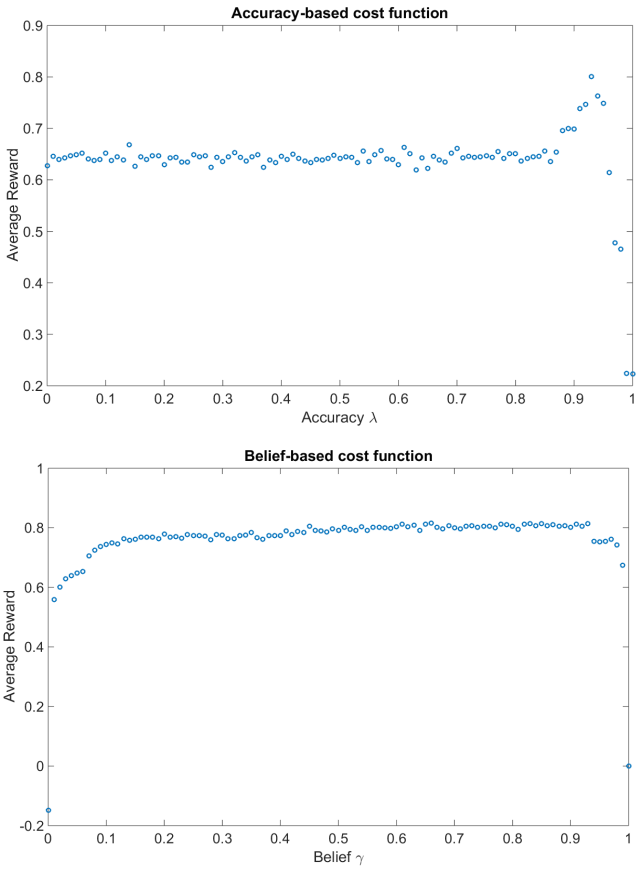

Figure 5: Plot of parameterized policies as a function of their respective parameter

Gradient ascent was used to verify the values found by the evolutionary algorithm. Table 3 provides the parameters used for the algorithm. Figure 5 shows the results of sweeping the threshold value over its possible range, and the average reward collected. Running gradient ascent on the two parameterized policies yielded the following optimal thresholds:

$$\lambda = 0.9310, \gamma = 0.8439.$$

Next, grammatical evolution was applied to the BorderCross problem. Figure 6 shows the results of running the two policy search methods (with and without reward shaping) on the BorderCross problem. As expected, the full policy search without shaping is not able to figure out how to successfully cooperate. The reward shaping method shows more potential as the

Table 3: Gradient Ascent Parameters

Parameter	Value
Episodes	10000
Error Tolerance	0.001
γ	0.2
ϵ	0.01
α_0	0.05
α_{t+1}	$\alpha_t \exp(-t/20)$

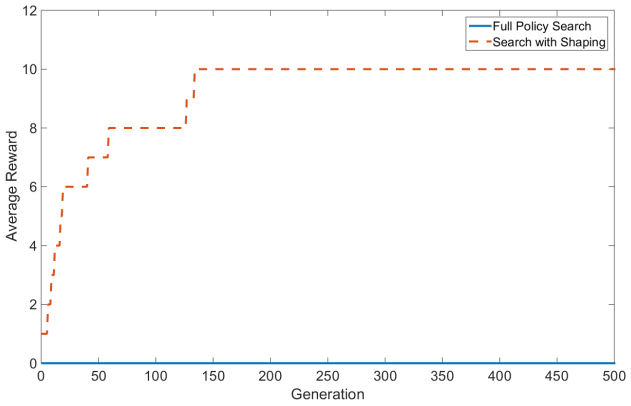

Figure 6: Results of running grammatical evolution on the BorderCross problem for the two different methods

intermediate rewards are accrued, which suggests that the algorithm is able to figure out some cooperation.

Figure 7 gives more insight into this, as it shows the two best policies generated at the end of the evolution. The policy without reward shaping appears to make random actions, which includes cooperating, but the robots are not in the correct locations to make it to the exit. There also doesn't appear to be any general strategy. However, the policy generated when using reward shaping is based on moving the robots to the same cell and then abusing the "cooperate" action. While this is an exploitation of the intermediate rewards (as in it doesn't lead the robots to the exit), it shows that reward shaping can help motivate the evolved algorithm to a successful solution. A different or modified version of this reward shaping strategy should be more effective in finding a successful strategy.

```
move(bc,1,"East")
move(bc,2,"North")
move(bc,2,"East")
cooperate(bc)
move(bc,2,"North")
move(bc,1,"North")
move(bc,1,"South")
move(bc,2,"North")
if close_enough(bc,1,9)
    move(bc,2,"East")
    cooperate(bc)
    move(bc,2,"East")
    move(bc,1,"East")
    cooperate(bc)
    move(bc,1,"South")
    move(bc,1,"South")
    move(bc,2,"North")
    move(bc,2,"East")
end
```

```
move(bc,1,"East")
move(bc,2,"North")
if close_enough(bc,2,7)
   move(bc,1,"South")
   move(bc,1,"South")
   move(bc,1,"East")
   cooperate(bc)
end
if close_enough(bc,2,0)
   cooperate(bc)
   move(bc,1,"East")
   move(bc,2,"East")
   cooperate(bc) [x3]
end
cooperate(bc) [x5]
```

Figure 7: Best generated policy by each of the policy search methods

6. Conclusion and Future Work

For the RockSample problem, the heuristic policies were able to achieve a much better performance than the full policy search, as expected. Nonetheless, compared to the POMDP baselines, the grammatical evolution algorithm was able to achieve comparable results after only 500 generations. The gradient ascent method also verified that the genetic algorithm was able to get the optimal threshold result (or close to it) for the parameterized policies. For this single-agent problem, the next step of this work is to combine gradient ascent with the grammatical evolution method, which was not completed due to time constraints for the project. In addition, creating more complex parameterized policies can provide insight into how well a hybrid method performs compared to other baselines.

For the BorderCross problem, a full policy search is not feasible due to the problem structure. Reward shaping shows promise, and some modifications could help better guide the search. However, the more interesting extension of this work is to quickly evolve a sub-optimal policy and then again use gradient ascent to improve the policy. Again, time constraints restricted the amount of work possible on this part.

For both problems, only open-loop policies (scripts) were created, while a more interesting policy is one that maps states to actions. One possible approach to this is to evolve a script for each state in the problems, and then evaluate the complete policy in the fitness function. In addition, the simple forward difference approximation for the gradient is low-order and inefficient. A better approach would be to use auto-

differentiation, which uses forward-accumulation and the chain rule to calculate the gradient of a function without the analytical form. Finally, evolved policies should include variables, and not constants, in the terminal set for gradient ascent to be applied properly.

All of these modifications of this work will made to better characterize hybrid methods involving grammatical evolution and gradient ascent. Given the results presented in this paper, it seems possible to augment grammatical evolution for better performance. As the main strength of grammatical evolution is the ability to have multiple levels of code abstraction, addressing the weakness of long policy searches should provide a powerful method for solving reinforcement learning problems.

7. References

[1] Kaelbling, Leslie Pack, Michael L. Littman, and Andrew W. Moore. "Reinforcement learning: A survey." Journal of artificial intelligence research(1996): 237-285.

[2] Hansen, Eric A. "Solving POMDPs by searching in policy space." Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence. Morgan Kaufmann Publishers Inc., 1998.

[3] Moriarty, David E., Alan C. Schultz, and John J. Grefenstette. "Evolutionary algorithms for reinforcement learning." J. Artif. Intell. Res.(JAIR) 11 (1999): 241-276.

[4] Whiteson, Shimon. "Evolutionary computation for reinforcement learning." Reinforcement Learning. Springer Berlin Heidelberg, 2012. 325-355.

[5] Beigi, Akram, et al. "Improving reinforcement learning agents using genetic algorithms." Active Media Technology. Springer Berlin Heidelberg, 2010. 330-337.

[6] Silver, David, and Joel Veness. "Monte-Carlo planning in large POMDPs." Advances in neural information processing systems. 2010.

[7] O'Neill, Michael, and Conor Ryan. "Grammatical Evolution."IEEE Transactions on Evolutionary Computation. 5 (2001): 349-358.