Evolutionary Reinforcement Learning for Sparse Rewards

Shibei Zhu[*]
Imperial College London
London, United Kingdom
shibei.zhu19@imperial.ac.uk

Francesco Belardinelli
Imperial College London
London, United Kingdom
francesco.belardinelli@imperial.ac.uk

Borja González León
Imperial College London
London, United Kingdom
b.gonzalez-leon19@imperial.ac.uk

ABSTRACT

Temporal logic (TL) is an expressive way of specifying complex goals in reinforcement learning (RL), which facilitates the design of reward functions. However, the combination of these two techniques is prone to generate sparse rewards, which might hinder the learning process. Evolutionary algorithms (EAs) hold promise in tackling this problem by encouraging the diversification of policies through exploration in the parameter space. In this paper, we present *GEATL*, the first hybrid on-policy evolutionary-based algorithm that combines the advantages of gradient learning in deep RL with the exploration ability of evolutionary algorithms, in order to solve the sparse reward problem pertaining to TL specifications. We test our approach in a delayed reward scenario. Differently from previous baselines combining RL and TL, we show that *GEATL* is able to tackle complex TL specifications even in sparse-reward settings.

CCS CONCEPTS

• **Computing methodologies → Planning with abstraction and generalization**.

KEYWORDS

deep reinforcement learning, temporal logic, evolutionary algorithm, sparse reward

ACM Reference Format:
Shibei Zhu, Francesco Belardinelli, and Borja González León. 2021. Evolutionary Reinforcement Learning for Sparse Rewards. In *Proceedings of the Genetic and Evolutionary Computation Conference 2021 (GECCO '21)*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3449726.3463142

1 INTRODUCTION

Reinforcement learning (RL) is an area of artificial intelligence where a learning agent interacts with a given environment to maximise her reward while taking a sequence of actions during her interaction [22]. In recent years, deep reinforcement learning (DRL), which integrates deep neural networks into RL, has emerged as a novel and effective method to solve RL problems, which has shown promising results, especially in the game-related domains [17, 19, 24]. In most games, the mechanism of reward (or reward function) is often part of the game setting (i.e., rules of chess or Go) that can be easily established. However, this is not the case in several other domains, where the reward can rarely be defined unequivocally. Moreover, in real-world scenarios, the reward function often does not obey the *markovian assumption* (i.e., the reward depends only on the agent's last action and last state), which is a fundamental feature that guarantees the convergence of several solving methods for RL problems. The use of temporal logic (TL) for goal specification in RL is attracting growing interest in recent years [5, 6, 23]. The expressive power of TL allows us not only to define the reward in a precise manner but also to deal with non-markovian rewards. That is, it allows the reward to depend on a sequence of past events besides the present state and action. This non-markovian feature of tasks is often very common to be observed in practical applications. For instance, if our goal is to prepare dinner, first we need to get all the ingredients, and then prepare the food by following an exact sequence of steps according to the cooking recipe.

Unfortunately, temporally-defined goals often give rise to sparse rewards, e.g., when the sequence of intermediate tasks tends to be long. As the probability of finding an exact sequence of actions decreases, and the length of its time horizon increases [10], it typically causes sparse feedback if the reward is given only when the whole specification is satisfied. This scarce feedback can potentially cause the agent to fail to learn the task at hand.

Contribution. In this paper, we tackle complex TL instructions where the agent has scarce feedback to learn from. We propose a novel solution to learn from sparse rewards generated from *co-safe* LTL specifications by using an evolutionary algorithm to perform the parameter space exploration. The EA method we develop is inspired by the hybrid genetic algorithms (GA) proposed in [12] that combines GA with off-policy policy gradient networks. Here we propose to use a new combination of GA with on-policy gradient methods that uses a different synchronisation scheme between them that can be applied for both discrete and continuous action space. We are interested in the on-policy methods since this may be more stable in practice (as it learns from experience sampled from its own policy rather than using experience generated by other policies) compare to its off-policy counterpart. And, unlike the off-policy methods, on-policy methods do not need to use experience replay buffer.

To the best of our knowledge, our *Gradient-Evolutionary Algorithm with Temporal Logic (GEATL)* is the first algorithm to combine temporal logic, (deep) reinforcement learning, and evolutionary algorithms to allow agents to learn complex, temporally-extended tasks. In Sec. 4, we show that GEATL can achieve competitive results with TL specifications for both sequential and interleaving

[*]Now at Aalto University, shibei.zhu@aalto.fi

GECCO '21, July 10–14, 2021, Lille, France
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463142

tasks proposed by earlier works [1, 23], even in scenarios with sparse feedback (see Sec. 4.1), where previous approaches struggle.

Related work. While there is a wealth of contributions combining RL and TL [14, 15, 23], they mainly focus on integrating increasingly expressive logical languages or on convergence proofs for non-Markovian rewards [7, 23]. The inefficient learning ability of agents due to sparsity of the temporally-extended reward is partially addressed in [4], where the TL system is enhanced with additional search techniques, in this case, reward shaping for MDP planners, such as PROST [11].

The underlying issue of sparse rewards is sometimes referred to as *hard-exploration problem* [8]. Previous approaches integrating TL specifications into RL [6, 9, 14] rely on continuous access to a global oracle, typically a labelling function, that provides feedback on the progress of the specification to the RL agent. In Sec. 4.2 we discuss that similar approaches struggle when facing environments where the feedback from the oracle is delayed to the end of an episode. Amongst the wealth of techniques intended to tackle sparse feedback, of particular relevance for the present work are [12, 21]. In [21] genetic algorithms (GA) are combined with novelty search to deal with tasks that have sparse or deceptive reward functions, which however require to detect the novelty in the action space given the environment. In [12], GA is combined alongside with gradient learning that uses an off-policy method with experience replay, which is more immediate to implement by using gradient learning.

2 BACKGROUND

2.1 Reinforcement Learning

Problems in reinforcement learning are typically modelled as Markov decision processes (MDP), which are tuples $M = \langle S, A, R, T, \gamma \rangle$, where S is the finite set of *states* of the system, A is the set of *actions* available to the agent, $T : S \times A \times S' \rightarrow [0, 1]$ is the *transition probability function*, $R : S \times A \times S \rightarrow \mathbb{R}$ is the *reward function*, and $\gamma \in [0, 1]$ is the *discount factor*. The learning process of the agent is based on the principle of trial-and-error [22]: at each time-step, the agent receives a reward from the environment, while finding the optimal policy $\pi^* : S \rightarrow A$ that maximises the discounted future reward or discounted return. This return is defined at time t as $G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$, where R_{t+k+1} is the reward at time $t + k + 1$.

Deep reinforcement learning (DRL)makes use of deep neural networks (DNN) as function approximator and incorporates *replay buffer*, a memory buffer to enable the reuse of experiences generated by the agent for the training process of the DNN. Several approaches in DRL have been proposed, which can be classified as *value-based* (or *critic-only*), which approximate the value function by using gradient descent, or *policy-based* (or *actor-only*), which use gradient ascent to estimate the policy directly by maximising the overall return. In this paper, we adopt a third category, known as *actor-critic*, which uses both value function and policy approximator.

2.2 Linear Temporal Logic

Linear temporal logic (LTL) is a propositional modal logic originally introduced to reason about reactive systems [18]. Amongst the variants of LTL, *co-safe* LTL and *LTL over finite traces* (LTL_f) have been successfully applied to specify agent's goals in reinforcement learning [5, 6, 23]. In this paper, we adopt *co-safe* LTL, which is a fragment of LTL with a restricted syntax to ensure deciding the truth of formulae in a finite number of steps. The formulae of *co-safe* LTL are built according to the following grammar [13]:

$$\varphi ::= p \mid \neg p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \varphi U \varphi$$

where $p \in \mathcal{AP}$ is a propositional atom.

2.3 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are black-box optimisation techniques inspired by Darwin's principle of natural selection [2]. Its application to neural networks is being often referred to as *neuroevolution* [20] as it explores the parameter space of neural networks (i.e., weights and bias matrices). Given an *objective function* (sometimes referred to as *fitness function*) to be optimised, and a set of randomly initialised solutions, where the set is also known as *population* and each of its elements as an *individual*, EA iteratively updates or "evolves" this set towards a possibly better solution space, by using the processes of *selection*, *mutation* and *recombination* (aka *crossover*) [3]. As evolution favours those individuals that best adapt to the environment, the selection operator is always biased toward individuals with higher fitness to produce the next generation of solutions.

3 EVOLUTIONARY REINFORCEMENT LEARNING

In this work, we put forward a solution to alleviate the learning inefficiency problem caused by sparse reward signal under a delayed reward scenario (see Sec. 4.1) using temporal logic as reward function. This sparsity of feedback can disorientate the agent's decision as no immediate learning guidance is given. We use an evolutionary algorithm to address this issue as it does not need immediate feedback, but rather than a fitness (or quality) measurement for the performance to optimise the final solution. We use an elite-based exploration in the parameter space that can potentially deal with hard-exploration scenarios by perturbing the parameters space to encourage diversity of actions. However, the convergence speed of EAs with large parameter space can be slow. Thus, we use gradient-learning to speed-up the learning process of the evolution-based agent. To this end, we first define the representation of the EA population and the architecture for the network of gradient learning, and then the integration of these two mechanisms.

EAs encourage more diverse policies for long-term goals by perturbing the parameter space (i.e., weight matrices of the neural networks). This motivates for the individuals of our population to be policy-based rather than value-based, as perturbations in the parameter space for the latter do not necessarily encourage more diverse actions. Specifically, value-based networks approximate the value function that calculates the Q-value for each state-action pair, and then a greedy policy simply selects the action with the highest Q-value. Hence, perturbations in value-based networks do not necessarily lead to more diverse policies, but rather introduce randomness in the value function prediction that is acting as a proxy when choosing actions. Thus, we use a different policy network

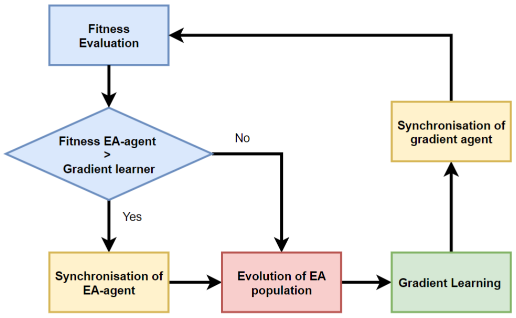
Fitness Evaluation
Fitness EA-agent > Gradient learner
No
Yes
Synchronisation of gradient agent
Synchronisation of EA-agent
Evolution of EA population
Gradient Learning

Figure 1: Workflow of GEATL

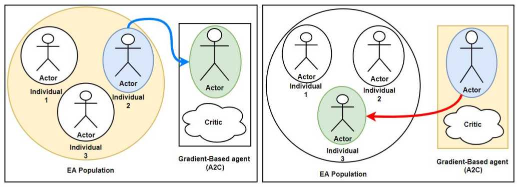

Figure 2: Synchronisations between the EA-based agent and the gradient-based agent.

(also known as actor networks) to represent each individual within the population.

The workflow of EA solutions is comprised of the processes of fitness evaluation and evolution of the solution (selection, mutation, and crossover). We integrate gradient learning by adding a synchronisation step between the EA population and the gradient learner. We refer to the process of integrating the gradient learner with the EA policies as *synchronisation* that is comprised by 2 types which are detailed hereafter. Our solution applies EA as the main learning framework, and uses gradient learning to improve the solution generated by the EA. The workflow of the proposed solution is outlined in Fig. 1, and details about the procedure in Alg. 1.

Synchronisation of the gradient-based learner. This process aims to guide the learning process of the gradient-based agent by inserting the policy network of the fittest individual of the population in the gradient-based learner to accelerate the convergence, especially when it encounters saddle points. Given the actor network $\pi_{\theta_{rl}}$ of the gradient-based learner parameterised by θ_{rl}, and the best individual of the population represented by an actor network $\pi_{\theta_{ea}}$ parameterised by θ_{ea}. The resulting $\pi_{\theta_{rl}}$ is defined as:

$$\pi_{\theta_{rl}} = \begin{cases} \pi_{\theta_{ea}} & \text{if } fitness(\pi_{\theta_{rl}}) < fitness(\pi_{\theta_{ea}}); \\ \pi_{\theta_{ea}} & \text{if } fitness(\pi_{\theta_{rl}}) = fitness(\pi_{\theta_{ea}}) \text{ and } prob < 0.5; \\ \pi_{\theta_{rl}} & \text{otherwise.} \end{cases} \tag{1}$$

Where the fitness function is the episodic return of the agent which is detailed in Sec. 3. In Eq. 1, when the EA-based actor has the same performance of the gradient-based learner, we still have a 50% chance of synchronising the actor of the EA solution into the gradient-based learner to encourage exploration with gradient learning as well.

Algorithm 1: GEATL

```
Initialise the gradient-based learner $rl_\theta$ with actor $\pi_\theta$ and
  critic $Q_\theta$ with weights $\theta$.
Initialise a population $pop_\pi$ of $k$ actors.
Define random number generator $r() \in [0,1]$, number of
  generation $x$ synchronisation period $\omega$.
for generation = 1, x do
    for actor $\pi \in pop_\pi$ do
        fitness = Fitness_evaluation($\pi$)
    end
    $fitness_{rl}$ = Fitness_evaluation( $rl_\theta$ )
    Select the individual with $max(fitnesses_{pop_\pi})$
    if $max(fitnesses_{pop_\pi}) > fitness_{rl}$ then
        Synchronisation($pop_\pi, rl_\pi$)
    end
    Evolution_EA_population($pop_\pi$)
    Update $rl_\theta$ using
     $\nabla_{\theta'} log\pi_{\theta'}(a_t|s_t)A_{\theta,\theta'}(s_t,a_t) + \beta\nabla_{\theta'}H(\pi_{\theta'}(s_t))$
    if generation mod $\omega$ = 0 then
        Synchronisation($rl_\pi, pop_\pi$)
    end
end
```

Algorithm 2: Evolution EA population

```
Select the first $e$ individuals $\pi \in pop_\pi$ based in fitness as
  elites where $e = \text{int}(\psi * k)$
Select $(k - e)$ actors $\pi$ from $pop_{pi}$, to form the set S using
  tournament selection with replacement
while $|S| < (k - e)$ do
    $\pi_1, \pi_2$ = tournament_selection($\pi_e$)
    $\pi_{new}$ = Crossover($\pi_1, \pi_2$)
    Append $\pi_{new}$ to $S$
end
for Actor $\pi \in S$ do
    if r() < $mut_{prob}$ then
        Mutate($\pi$)
    end
end
```

Synchronisation of the EA-based learner. This process is performed at each generation after the gradient learning of our gradient-based learner. The actor of the gradient-based agent is inserted to replace the worst-performing individual of the EA population, which improves the overall performance of the whole population. Note that the critic (which is not present in the EA population) remains unaltered throughout this process so that the gradient learner retains some of its previous knowledge even when its actor is updated. Given $\pi_{\theta_{rl}}$ as the parameter vector of the gradient-based learner parameterised by θ_{rl}, and the worst individual of the population represented by an actor network $\pi_{\theta_{ea}}$ parameterised by θ_{ea}. The resulting $\pi_{\theta_{ea}}$ is defined as: $\pi_{\theta_{ea}} = \pi_{\theta_{rl}}$.

Fitness evaluation. The performance of the individuals in the EA population is evaluated by the *fitness function* under a time-limited (i.e., one episode) scenario,this uses the same reward function for the standard RL agent with the exception that its (episodic) value is given at the end of the episode. Evolution defines how EA iteratively generate (or evolve toward) a better solution space based on the fitness score and the current population. This process comprises of*selection*, *mutation*, and *crossover*, which are probabilistic operations that introduce more randomness into the new solution. The selection method used to choose parents is *tournament selection with replacement* [16].

Mutation. Given a population of individuals where each individual is a policy network parameterised by parameter θ that outputs policy π_θ, the mutation of individuals is a probabilistic operation that perturbs the parameter vector $\widetilde{\theta}$. Different criteria can be used, here we used the following appearing in [12]:

$$\widetilde{\theta} = \begin{cases} \theta + \mathcal{N}(0, \theta * super\_mut\_stregth) & \text{if prob} < supermut\_prob; \\ \mathcal{N}(0,1) & \text{elif prob} < reset\_prob; \\ \theta + \mathcal{N}(0, \theta * mut\_stregth) & \text{otherwise.} \end{cases} \tag{2}$$

where $super\_mut\_stregth$ set to 10 and $supermut\_prob \in [0,1]$ are the super mutation probability and the mutation rate respectively, which determine the Gaussian noise to introduce, and $reset\_prob \in [0,1]$ is the probability of resetting the parameter into a normal distribution. Lastly, the normal mutation probability and rate depend on $mut\_stregth$ that is set to 0.1.

Crossover. Given two parent parameters vectors θ_a and θ_b, the process of crossover defines the resulting parameter vector $\widetilde{\theta}$ as the concatenation of the parent vectors as follow:

$$\widetilde{\theta} = \begin{cases} concact(\theta_a[0,\alpha],\ \theta_b[\alpha,n]) & \text{if prob} < 0.5 \\ concact(\theta_b[0,\alpha],\ \theta_a[\alpha,n]) & \text{otherwise} \end{cases} \tag{3}$$

where n is the length of the parameter vector and $\alpha \in [0,n]$ is generated randomly to determine the exact point of intersection.

4 EXPERIMENTAL EVALUATION

The environment used for the experiments is a Minecraft-inspired 2D grid world proposed in [1], as modified in [23]. The agent can interact with different objects that are distributed on a finite size map. Potential tasks include mining raw materials (i.e., iron, wood, and grass), combining different material in a specific order to create new objects or tools. Detailed description is given in the supplementary material. The experiments were run on a machine with Intel i7-7700k as CPU and Nvidia GTX1080 as GPU.

4.1 Reward Distribution

The reward is similar to the one used in previous works of TL and RL [6, 23], where the agent has access to a *labelling function*, which provides feedback on the progression of the specification. However, we evaluate harder scenarios, where the feedback of the labelling function is available only at the end of the episode. That is, at every interaction the agent only receives standard negative feedback for the steps it takes, and only at the end of the episode, the labelling function provides the positive feedback as the result of completing the tasks. We refer to this setting as *delayed rewards*, where the agent is forced to solve the problem of *temporal credit assignment*, i.e., to solve the problem where the agent needs to discover which actions contribute the most to the final reward.

n°tasks	*GA*	*A2C*	*LPOPL*	*ERL*	*GEATL*
2	-6.5(5.8)	-6.5(5.8)	-6.5(5.8)	-6.5(5.8)	-6.5(5.8)
3	-98(0.5)	-84.33(14.3)	-83.67(1)	-12(2.5)	-12.5(2.5)
4	-99(0)	-98(1)	-98 (1)	-36.75(50.4)	-15(28.1)
5	-300(0)	-300 (0)	-300 (0)	-16(28.4)	-17(20.7)

Table 1: Episodic reward obtained for sequential tasks. For each group, we give the median with its corresponding IQR. The groups with tasks from 2 to 4 have the worst value as -100 where the group of 5 has it worst value as -300 (which means that the agent has 100/300 steps per episode respectively).

n°tasks	*GA*	*A2C*	*LPOPL*	*ERL*	*GEATL*
2	-6.5(5.8)	-6.5(5.8)	-6.5(5.8)	-6.5(5.8)	-6.5(5.8)
4	-98(0.5)	-84.33(14.3)	-83.67(1)	-8.8(0.2)	-9.6(1.6)
5	-98(0.5)	-84.33(14.3)	-83.67(1)	-30.2(15.8)	-24.42(10.4)
6	-300(0)	-300(0)	-300(0)	-297(3)	-23(2)
7	-300(0)	-298(0)	-299(1)	-27 (136)	-27(23)

Table 2: Episodic reward obtained for interleaving tasks. The groups with tasks from 2 to 5 have the worst value as -100 while the tasks of 6 to 7 has their worst value as -300.

4.2 Experimental Setting

We now compare the result of EA (specifically Genetic Algorithm (GA)), A2C, LPOPL, GEATL and ERL algorithms in dealing with the experimental setting here proposed. The setting of these approaches is detailed in the supplementary material attached (Sec.3). The results are shown on Tab 1 and Tab 2.

As different length of specifications have a different level of difficulty and the total number of specifications is rather huge, we group and measure the performance of the specifications according to its number of tasks, i.e., all the specifications with 2 tasks belong to the same group. We use the median (also known as the 50th percentile) that is not skewed by atypical data (i.e. extremely high or low values) as the main measurement to show the central tendency of the results. We also give the interquartile range (IQR) to visualise the dispersion amongst the result of the same group. Both median and IQR are calculated using the average reward for each specification in 3 independent training processes. The IQR is given by subtracting the 75th percentile with the 25th percentile. For the groups that only have one specification per group, in this case, only the specification of interleaving with 6 and 7 tasks, we use the results of 3 different run to calculate their median and IQR. The reason of having only one specification per group is due to the time limitation we had.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we introduced GEATL, a gradient-evolutionary RL algorithm to tackle the problem of sparse rewards caused by temporal specified goals. Previous works on RL-TL show promising approaches to solve complex specifications [1, 6, 9, 23]. However, these works rely on online access to a "global oracle" called *labelling function* to guide the learning process of their agents. Our experiments show that when such guidance, i.e., the positive rewards from the labelling function, is given only at the end of the episode, those agents may fail to solve the proposed specification.

REFERENCES

[1] Jacob Andreas, Dan Klein, and Sergey Levine. 2017. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*. 166–175.

[2] Thomas Back. 1996. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press.

[3] Thomas Bäck and Hans-Paul Schwefel. 1993. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation* 1, 1 (1993), 1–23.

[4] Alberto Camacho, Oscar Chen, Scott Sanner, and Sheila A McIlraith. 2017. Non-markovian rewards expressed in LTL: guiding search via reward shaping. In *Tenth Annual Symposium on Combinatorial Search*.

[5] Alberto Camacho, R Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2019. LTL and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*. 6065–6073.

[6] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. 2019. Foundations for restraining bolts: Reinforcement learning with LTLf/LDLf restraining specifications. In *Proceedings of the International Conference on Automated Planning and Scheduling*, Vol. 29. 128–136.

[7] Giuseppe De Giacomo and Moshe Y Vardi. 2013. Linear temporal logic and linear dynamic logic on finite traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*.

[8] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2019. Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995* (2019).

[9] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. 2018. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*. PMLR, 2107–2116.

[10] Sham Kakade and John Langford. 2002. Approximately optimal approximate reinforcement learning. In *ICML*, Vol. 2. 267–274.

[11] Thomas Keller and Patrick Eyerich. 2012. PROST: Probabilistic Planning Based on UCT.. In *ICAPS*. 119–127.

[12] Shauharda Khadka and Kagan Tumer. 2018. Evolution-guided policy gradient in reinforcement learning. In *Advances in Neural Information Processing Systems*. 1188–1200.

[13] Orna Kupferman and Moshe Y Vardi. 2001. Model checking of safety properties. *Formal Methods in System Design* 19, 3 (2001), 291–314.

[14] Borja G León and Francesco Belardinelli. 2020. Extended Markov Games to Learn Multiple Tasks in Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2002.06000* (2020).

[15] Xiao Li, Cristian-Ioan Vasile, and Calin Belta. 2017. Reinforcement learning with temporal logic rewards. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 3834–3839.

[16] Brad L Miller, David E Goldberg, et al. 1995. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems* 9, 3 (1995), 193–212.

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[18] Amir Pnueli. 1977. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. IEEE, 46–57.

[19] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

[20] Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127.

[21] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2017. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567* (2017).

[22] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

[23] Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. 2018. Teaching multiple tasks to an RL agent using LTL. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 452–461.

[24] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.