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

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 EXPERIMENTAL ENVIRONMENT

The environment used for the experiments is the 2D map in the Fig 1, where the agent is represented by the capital letter *A*, the interactive objects within the environments are given as lower case letter *a, b, c, d, e, f, g, h*, and the *X* represents the border of the map. This map is shown is Fig. 1.

1.1 Action and Observation space

Each agent has a set of actions comprised of all the possible movements within the world *A = {Up, Down, Left, Right, Wait}*. The illegal action (i.e. move to the outside of the world) will be interpreted as *wait*.

The state space is a discrete grid, where the feature vector is given as the manhattan distance between the agent and each object on the map. That is, given an environment with k objects, we will have a state vector of length k with each element of it as a distance measurement between one object y and the agent A. Mathematically this distance is given as $D(A, y) = \sum_{i=0}^{n-1} |A_i - y_i|$, where A and y here are the position vector (e.g., x and y in Cartesian coordinates) of the agent and the object respectively.

2 SPECIFICATIONS

The experiments consist of 2 set of different specification types: *sequential* [1] and *interleaving* [23]. The former requires the agent to perform the tasks in a specific order while the second has a more flexible order. The original sets are proposed in [1, 23], which are comprised of ten specifications of different length for each category. Here, we include 8 additional specifications for the sequential specifications, and the interleaving specifications remain the same.

The atomic events or properties that the agent can trigger in the environment are described by set $P = \{got\_wood, got\_grass, used\_factory, used\_toolshed, got\_iron, used\_workbench, used\_axe, used\_bridge\}$. On top of these, we build specifications in *co-safe* LTL. We define different goals for the agent based on the tasks to perform, where each task is associate with one event. For instance, the following formula intuitively specifies the construction of a lever in *co-safe* LTL with sequential specification:

$$\varphi_{lever} = \diamond(got\_wood \land (\diamond(got\_iron \land \diamond used\_factory)))$$

Some specifications can be also specified in an interleaving manner. For instance, in formula φ_{lever} the order of collecting wood and iron is fixed, but this order can be relaxed to first collecting either wood or iron:

$$\varphi_{collection} = \diamond(got\_wood) \land \diamond(got\_iron)$$

Informally, we say that formula φ_{lever} has length 3, because it involves three different tasks, at three different steps. The full specifications set can be ranged from length two to seven. For each specification, we train one network except for LPOPL where each task in the specification requires one network and the final specification is solved jointly by the equivalent amount of networks as the length of the specification. The full details of these specifications are included in the supplementary material (Sec. 2).

2.1 Sequential specifications

The original set of sequential specifications [1] is comprised by four specifications of length two (which involve two tasks), one specification of length three, four specifications of length four and one specification of length five.

The specification are given in the format of $\varphi = \diamond(a \land (\diamond b))$, for the simplicity we will list it as *ab*. Following by the same logic, the inital set is: *{ab, ac, de, db, fae, abdc, acfb, acfc, faeg, acfbh}*. We include also the set: *{acf, defe, fcac, dbca, aefe, acfba, acfca, fbacb}*.

2.2 Interleaving specifications

In this setting, we use the interleaving specifications proposed in [23]. The whole set is comprised by four specifications of length two, two specifications of length four, two specifications of length five, one specification of length six, and one specification of seven.

The specification is given in the format of $\varphi = \diamond ab \land \diamond bc$, where the order of performing *ab* and *bc* is not a requirement but rather flexible. As for the previous case, we will list it as *ab-bc* for simplicity. Following by the same logic, this set is made of: *{ab, ac, de, db, ae-fe, dc-abc, fb-acb, fc-ac, fbh-acbh, aeg-feg}*.

3 HYPER-PARAMETER SETTING

While LPOPL uses one network per task within the specification, all the other approaches use one network per specification. The implementation of A2C uses the independent actor and critic networks with 3 layers and 128 neurons for each network. The hidden layers have ReLU activation, and we use softmax for the actor-network

[*] 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

```
XXXXXXXXXXXXXXXXXXXXX
X                   X
X        c          X
X  g   d   a f      X
X           f       X
X                   X
X                   X
X                   X
X    g              X
X a                 X
X   fd    A         X
X                   X
X                   X
X         b   b     X
X          a        X
X                f  X
X   h      ac     d X
X    a          e   X
Xd             hd   X
X      ef           X
XXXXXXXXXXXXXXXXXXXXX
```

Figure 1: Environment map for the experiment

and ReLU for the critic. We use an entropy term of 0.01 and a learning rate of 0.0005. The setting of LPOPL is the same as proposed in [23], but the reward is given to all the networks only at the end. As mentioned in the main text, this reward system reduces the performance, but in practice, this method has lesser restrictions for real-world application, i.e., we do not need online access a global oracle. The pure genetic algorithm has the same setting as the EA part of GEATL which is specified in the Sec. 3 of the main text. For ERL [12], we use the new discrete action version released by the authors as the original version is only available for the continuous action space. All the setting remain the same as proposed by the author except the number of populations and the size of the batch are reduced to 10 and 64 respectively. In GEATL, the gradient-based learner uses the same setting as the standard A2C [?], and at each generation, we train it using 10^4 steps. For all the EA-approaches, we use a population of 10 individuals.

For the specifications that use 300 episodic steps during training, we train the A2C for 10^6 steps in total, while for the ones that use 500 episodic steps, we use 10^8 steps. Same apply for the LPOPL. For the EA-based approaches, we use instead number of evolutions to better measure the performance as we discussed in the main text. For the sequential setting that uses 300 training episodic steps, we use 500 evolutions, while for the case of 500 training episodic steps, we use 1000 evolutions. For interleaving set, that we remain 500 evolutions for both 300 and 500 episodic cases.

4 EXPERIMENTAL RESULTS

This section shows the performance of GEATL during the training phase. In the main document, we show the final result in terms of median and its corresponding IQR for each group of specifications (where the same group members have the same length of specifications). Here, we show the test score using the median, the 25th percentile and the 75th percentile for each group. These results are obtained after every generation of training.

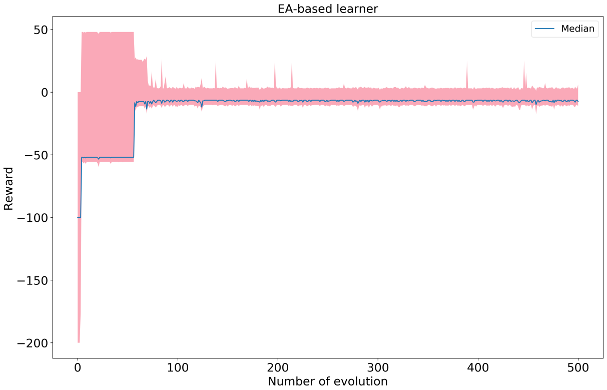

Figure 2: Episodic reward for the EA-based population for sequential specifications of length 2.

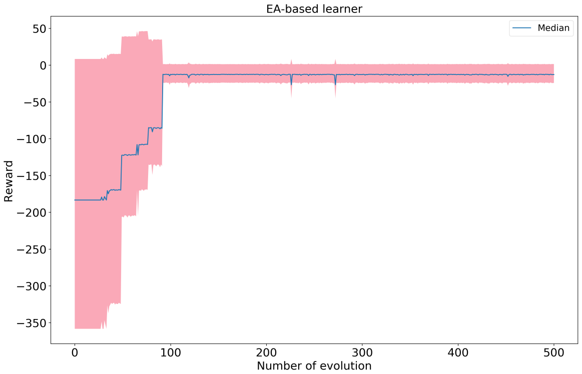

Figure 3: Episodic reward for the EA-based population for sequential specifications of length 3.

4.0.1 Sequential specifications. Under this category, we show the results of 4 different groups: specifications of length two (Fig. 2), length three (Fig. 3), length four (Fig. 4) and length five (Fig. 5).

From the graphs, we can observe that the specifications with a lesser number of tasks in involved have a smaller interquartile range. This is mainly caused by the fact that the specifications are easier to solve and the optimal average results (episodic reward) for the specifications of the same group are often similar which give a smaller interquartile range value.

4.0.2 Interleaving specifications. Under this category, we show the results of 5 different groups: specifications of length two (Fig. 6), length four (Fig. 7), length five (Fig. 8),length six (Fig. 9) and length seven (Fig. 10). As we are not imposing a strict order to performing the tasks within the specification, we can observe that the agent can solver specifications of longer length. As we are giving a lesser number of generation for harder setting (i.e. 500 episodic steps for training and 300 training step for testing), the convergence to the optimal results is slower and the interquartile range is wider than other groups.

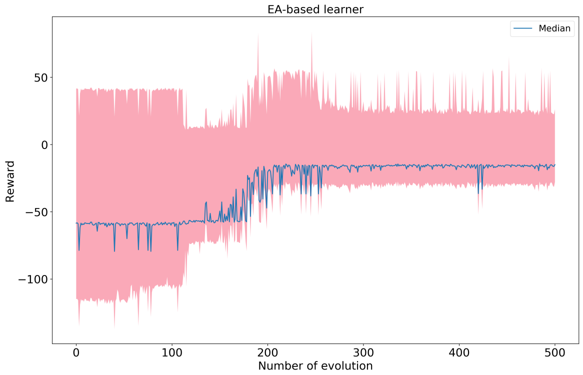

Figure 4: Episodic reward for the EA-based population for sequential specifications of length 4.

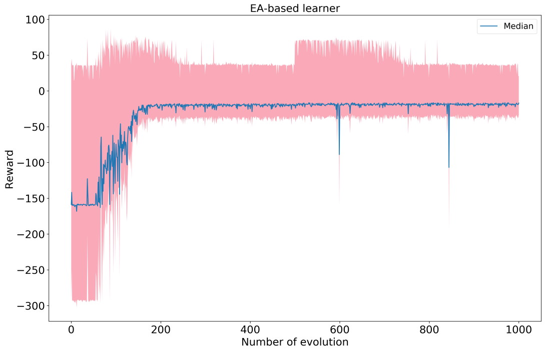

Figure 5: Episodic reward for the EA-based population for sequential specifications of length 5.

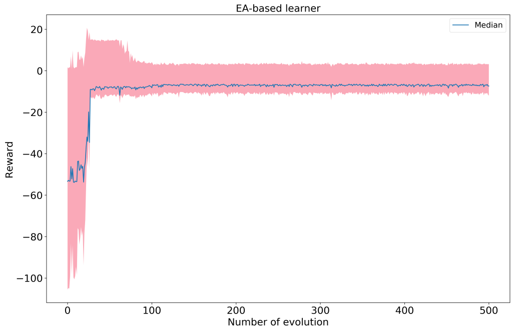

Figure 6: Episodic reward for the EA-based population for interleaving specifications of length 2.

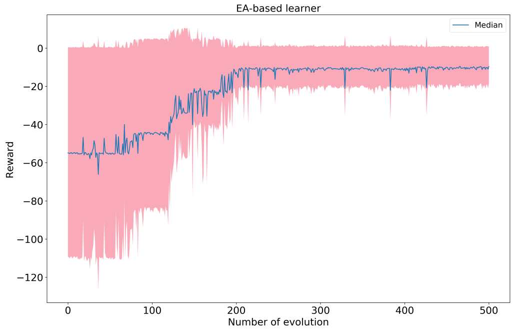

Figure 7: Episodic reward for the EA-based population for interleaving specifications of length 4.

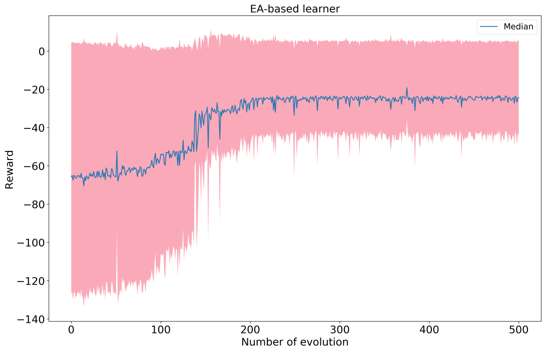

Figure 8: Episodic reward for the EA-based population for interleaving specifications of length 5.

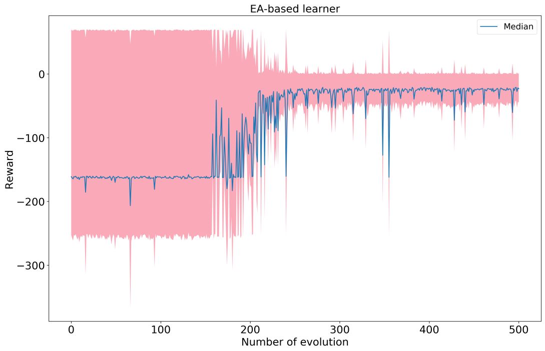

Figure 9: Episodic reward for the EA-based population for interleaving specifications of length 6.

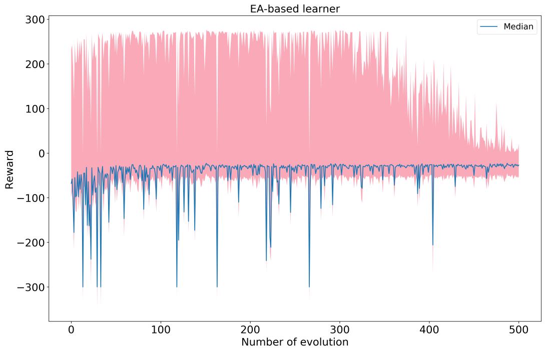

Figure 10: Episodic reward for the EA-based population for interleaving specifications of length 7.

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.