EXPLORING PAIRED OPEN ENDED TRAILBLAZER CS748 FINAL PROJECT REPORT

Arjit Jain, Yash Sharma & Aditya Vavre
Department of Computer Science & Engineering
Indian Institute of Technology Bombay
Mumbai, India
{170050010,17D070059,170050089}@iitb.ac.in

1 INTRODUCTION

"Give a Man a Fish, and You Feed Him for a Day. Teach a Man To Fish, and You Feed Him for a Lifetime"

Neural networks excel at solving tasks with properly defined objectives. What's the next step? What if the AI algorithm itself could generate novel complex problems, and then come up with agents to solve those problems? This open-ended problem solving process is exactly the subject of a recently proposed algorithm called POET (Wang et al., 2019) The Paired Open-Ended Trailblazer, or POET, relies on the coevolution of environments and agents. Maintaining a population of environments and agents allows for goal-switching, where the idea is that the solution to one environment might be a stepping stone to a new level of performance in another. After starting from an initial environment, POET loops forever, and in each loop, it performs three tasks: (1) **Mutation**: Mutation to generate new environments from current environments. (2) **Optimization**: Optimization of agents in their respective environments. (3) **Transfer**: Transfer of agents from one environment to another if favourable, also called goal switching.

The remarkable thing about POET was that it was able to find environments for which direct optimization, or even curriculum based optimization would fail necessitating the need of these stepping stones, and goal switching.

However POET lacks experimental benchmarking. To the best of our knowledge, POET has only been tested on the BiPedal Walker environment. The implementational choices made by POET, favoring novel complex environment generation for the BiPedal Walker, might not generalize to other tasks and domains, and hence necessitates testing POET on different environments.

Hence, in this work, we then aim to answer two fundamental questions about POET:

> **Validation**: Can the POET algorithm be extended to other environments?
>
> **Efficiency**: Can we find these novel environments faster?

Answering the above questions can help us apply POET intelligently to other domains. The applications of open-ended algorithms like POET are endless, but perhaps most significant in AI safety, with test-case generation, and stress testing.

Our contributions are as follows: We apply the POET framework on the Minigrid Environment, a widely used environment for benchmarking reinforcement learning algorithms. We discuss the challenges faced in extending POET from a continuous action space with dense rewards, to a discrete action space with sparse rewards. We show that the evolutionary strategy used as the optimization procedure to train agents in POET requires hand-engineered, low-dimensional state representations. We propose an evaluation metric that captures the dual optimization involved in POET, with respect to generating tougher environments, and training better agents. We also propose a modification to POET's mutation strategy that favours reproducing environments with higher participation in goal switching.

After suitably addressing the design choices mentioned above, even with very limited compute, we were able to show that POET can indeed generate novel complex environments, along with agents that outperform direct optimization on these environments.

A recent line of work investigates how neural networks use "shortcuts" to accomplish the task at hand (Geirhos et al., 2020). During our experiments, we experienced a similar shortcut learning phenomenon. We discuss how these shortcuts can arise in open-ended algorithms like POET, and propose a way to mitigate them.

Our code can be found at `https://github.com/ArjitJ/poet`

2 Related Work

POET has gained significant attraction from the research community. However, most of the works that POET inspired have been on setter-solver paradigms (Racaniere et al., 2020). The goal there is not novelty search, but finding a curriculum of tasks to perform well on an end goal (Fang et al., 2021) or on tasks with emergent complexity (Dennis et al., 2020). In fact, the only work, to the best of our knowledge, which uses POET is the enhanced POET (Wang et al., 2020) which was proposed to fix some of the inefficiencies of POET, and make it more domain independent. One of the reasons that we have identified for the lack of activity on the original POET work, is the implementation. While the authors release their code, we have found it to be buggy, hard to read, and overall too environment specific. As a result, one of our contributions through this project is to modify their implementation, and make it work for a wide range of environments to foster research on POET. And, as we stated, POET has not been explored with respect to it's optimization algorithm, mutation strategy or agent-transfer.

3 Why prior works failed?

One of our mutations is changing the environment on which POET is run. To this end, we intend to apply POET for novelty search in the MiniGrid gym environment (Chevalier-Boisvert et al., 2018). As shown in Figure 1 this environment, while being computationally inexpensive, captures enough complexity for interesting search, including sequential elements like lock and key, as well as a variety of obstacles in the form of multiple moving balls, and lava. This is also a well studied environment, and has popular baselines like PPO and actor critic implemented (lcswillems, 2021).

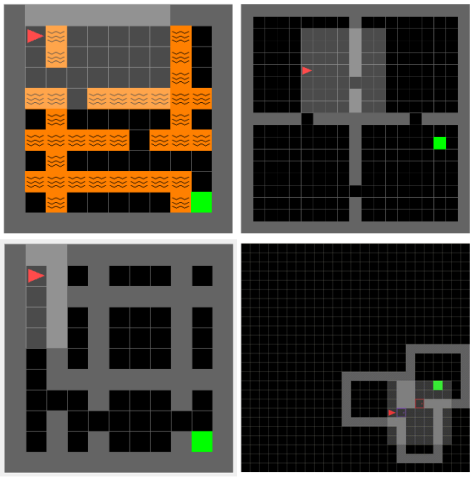

Figure 1: The MiniGrid environment space allows us to generate a wide range of diverse environments

Prior works (gdicker1, 2021) have also attempted to apply the POET algorithm on this environment, but were unsuccessful. We have identified the following possible reasons for failure:

- **State Representation**: POET, on the BiPedal Walker environment, used a hand-crafted semantically meaningful state representation. However, on the MiniGrid environment, we are using raw image pixels.

- **Sparse Rewards**: On the BiPedal Walker environment, the agent received immediate feedback at each step, whereas on our MiniGrid environment, we have sparse rewards. The agent receives reward only at the end of the episode, based on whether or not it completed the maze.
- **Action Space and Policy Network**: One of the reasons POET was able to use a simplistic optimization strategy like ES is because the action space was continuous. The outputs of the policy network could directly be used to control the motor joints of the agent. However, in our MiniGrid environment with discrete actions, the policy network outputs a probability distribution which is then used to sample the next action.

Identifying the possible failure cases of prior work paves the way for us to mitigate some of these issues. For instance, we could consider reward shaping, and hand-engineering the state representation.

4 BACKGROUND

4.1 POET

The Paired Open-Ended Trailblazer, or POET, relies on the co-evolution of environments and agents. The POET algorithm aims to enable an open-ended problem solving method. The algorithm repeatedly generates increasingly challenging environments and improves skills within a single run. It maintains a population of environments and a population of agents, and every environment is paired with an agent to make an environment-agent pair. This allows for agent transfer, where the idea is that the solution to one environment might be a stepping stone to a better performance in another environment.

The POET algorithm takes as input a simple starting environment $E^{init}(\cdot)$ and a randomly initialized weight vector θ^{init} (this represents the agent which can be in the form of a neural network). It maintains a list of environment-agent pairs $\mathrm{EA\_list}$. The POET main loop essentially performs three main steps in each iteration.

(1) It generates new environments(children) $E(\cdot)$ from current ones(parents) through mutation. POET randomly perturbs existing active environments to generate a new environment that is not too hard or not too easy. It does so while ensuring that the agents paired with the parent environments have been optimized enough to suggest that reproducing their respective environments would not be a waste of effort. Priority is given to those children environments that are most novel to facilitate the learning of more diverse problems in a single run.

(2) It optimizes each agent on its paired environment $(E(\cdot), \theta)$. The idea is that every agent in POET should be continually learning skills within its paired environment. POET uses Evolution Strategies as its optimization algorithm. The operations performed in one ES_STEP can be found in Algorithm 2.

(3) It attempts to transfer agents θ between environments. For example, if the paired agent θ^A in environment $E^A(\cdot)$ is stuck in a local optimum, one solution could be to transfer an agent θ^B from environment $E^B(\cdot)$. If θ^B performs better on $E^A(\cdot)$ then such a transfer is favourable and the performance on the former environment can be improved. POET attempts these transfers between active agents in each iteration.

This completes the main POET loop which is also formally described in detail in Algorithm 1. Through the above described steps, a single run of POET is able to generate a curriculum of increasingly challenging tasks, some of which benefit from the solutions of others.

5 METHODS

The Main Loop of the POET algorithm remains the same as described in Section 4.1. We do however propose one modification to the MUTATE_ENVS procedure. This is described here.

Algorithm 1: Main POET Loop

Input: Initial environment $E^{init}(\cdot)$ its paired agent denoted by policy parameter vector θ^{init}, learning rate α, noise standard deviation σ, iterations T, mutation interval N^{mutate}, transfer interval $N^{transfer}$

Initialize: *Set* EA_list *empty*;
Add $(E^{init}(\cdot), \theta^{init})$ to EA_list;
for $t = 0$ **to** $T - 1$ **do**
  if $t > 0$ **and** $t \mod N^{mutate} = 0$ **then**
    EA_list = MUTATE_ENVS(EA_list); // new environments created by mutation
  end
  M = len(EA_list);
  for $m = 0$ **to** M **do**
    $E^m(\cdot), \theta_t^m$ = EA_list$[m]$;
    $\theta_{t+1}^m = \theta_t^m$ + ES_STEP$(\theta_t^m, E^m(\cdot), \alpha, \sigma)$; // each agent independently optimized
  end
  for $m = 0$ **to** M **do**
    if $M > 1$ **and** $t \mod N^{mutate} = 0$ **then**
      θ^{top} = EVALUATE_CANDIDATES$(\theta_{t+1}^1, \ldots, \theta_{t+1}^{m-1}, \theta_{t+1}^{m+1}, \ldots, \theta_{t+1}^M, E^m(\cdot), \alpha, \sigma)$;
      if $E^m(\theta^{top}) > E^m(\theta_{t+1}^m)$ **then**
        $\theta_{t+1}^m = \theta^{top}$; // transfer attempts
      end
    end
    EA_list$[m]$ = $(E^m(\cdot), \theta_{t+1}^m)$
  end
end

Algorithm 2: ES_STEP

Input: An agent denoted by its policy parameter vector θ, an environment $E(\cdot)$, learning rate α, noise standard deviation σ

Sample $\epsilon_1, \epsilon_2, \ldots, \epsilon_n \sim \mathcal{N}(0, I)$;
Compute $E_i = E(\theta + \sigma\epsilon_i)$ for $i = 1, \ldots, n$;
Return: $\alpha \frac{1}{n\sigma} \sum_{i=1}^n E_i \epsilon_i$

5.1 Goal Switching focussed Mutation

As with other evolutionary algorithms, the mutation strategy is an important aspect of POET. POET starts with a list of parent environments. These parents then reproduce (by being perturbed randomly) to generate child environments. The child environments that are too easy or too hard are filtered out (Minimum Criterion). The remaining child environments are ranked by novelty and then admitted in the list of environments. If the number of environments exceed a certain maximum capacity then the oldest parent environments are pruned away.

At the heart of POET is goal-switching. Then why not modify the mutation strategy to increase the possibility of finding goal switching pairs? Let (E_{src}, A_{src}) and (E_{tgt}, A_{tgt}) be a goal switching pair. E_{src} is called a "stepping-stone" since an agent (A_{src}) trained on this environment (E_{src}) can outperform direct optimization (A_{tgt}) on some other environment (E_{tgt}). E_{tgt} is called a challenging environment since direct optimization on it is insufficient. We want to identify and study both stepping stones, as well as challenging environments.

We propose a small modification to POET's mutation strategy. When exceeding the maximum capacity, instead of removing the oldest parent environments, we propose to remove the parent environment that have had the least participation in goal-switching based transfers. An environment is said to participate in goal switching if it is either the source E_{src}, or the target E_{tgt}.

6 EXPERIMENTS

6.1 SETUP

We parameterize our environment with the probabilities of obstacles, and an obstacle level. We multiply the obstacle level by the size of the grid, and those are the maximum number of obstacles that can be placed. The obstacles are lava, box, ball, door, wall. Since our parameters consist of probabilities, even for a fixed parameter set, say [lava_prob, box_prob, ball_prob, door_prob, wall_prob, obstacle_level] = [0.5, 0.5, 0.5, 0.5, 0.5, 1] can potentially generate all possible environments. To create child environments, a parent can independently modify each probability by [+0.05, 0, -0.05], and the obstacle level by [0.33, 0, -0.33]. To calculate novelty, the distance between environments is calculated as the sum of the absolute differences between the different parameters. Implementation details can be found in the appendix.

6.2 EVALUATION METRICS

One straightforward measure of the agent's performance is the average reward. Hence, as the POET algorithm progresses, one can monitor the average reward over the different environments in the population. However, due to the rather conflicting goals of POET, a low average reward can indicate either tough environments, or poorly trained agents or both. Hence, we propose a **cross evaluation** of all environments against all agents.

Formally, we construct a $N \times N$ matrix S, where N is the number of environments in the population. S_{ij} contains the average reward obtained by agent A_i on environment E_j.

Apart from reward based measures, we also look at the number of goal switching based transfers, and the total number of novel complex environments generated.

6.3 OPTIMIZATION USING EVOLUTIONARY STRATEGIES

We found that a direct extension of POET to the minigrid environment failed to converge (gdicker1, 2021). In order to mitigate this, we made the following changes. We moved from a sparse reward setting, where the agent recieved a $0 - 1$ reward at the end of an episode based on whether or not it reached the goal state, to a dense reward setup where the agent recieved a -0.1 reward per timestep, with a maximum of 160 timesteps, and $+100$ reward on reaching the goal state. If the agent falls into lava, then the episode is terminated, and the agent receives a reward of -25. We also decreased the state space dimension by moving from a raw pixel based representation, required 3 values per grid cell, to an nominal object encoding based representation where each grid cell required only 1 value. The encoding dictionary can be found in the appendix. We used a neural network with 2 hidden layers, with the hidden dimension 16 and tanh activation. The POET algorithm, on the BiPedal Walker, started from the simplest environment, namely the completely horizontal track. Similarly, in our setup, we start from the simplest minigrid environment, namely the "empty" environment with only the goal state and no obstacles. However, the resulting neural network was still too large for ES to optimize, and as a result even after sufficient training iterations we did not notice any improvement in the agent's performance.

6.4 MINIMAL WORKING EXAMPLE

We designed an experiment to see if with an extremely low dimensional state space, and a small neural network, are we able to get ES to overfit on the empty environment. In this state representation, we only included the bare minimum needed to solve the empty environment, the coordinates of the agent's position and the direction of the agent. We used a neural network with 2 hidden layers, with the hidden dimension 8. With this configuration, POET with ES was able to achieve decent performance on the empty environment. We were able to improve the performance by changing the bias initialization of the softmax layer. We used a bias of $+1$ for the action forward, 0 for left and right, and -1 for obstacle related actions like pick up, drop and toggle.

To our surprise, we noticed that with this simple configuration, POET was able to generate, and solve, a variety of novel environments. Upon closer inspection, we noticed that most of these novel environments shared a path with no obstacles to the goal. The agents were able to exploit this, and

because of the minimum criterion thresholds, the environments which had obstacles on this path resulted in low rewards, and hence couldn't reproduce and were slowly wiped out. As a result, the population contained more and more of these environments with a this "shortcut", and this reinforced the exploitation by the agents.

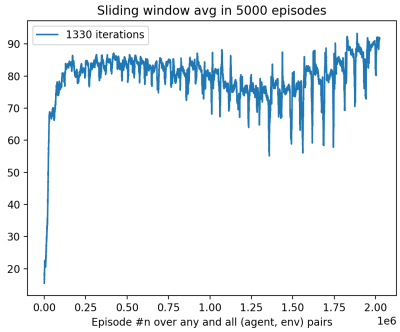

Figure 2: Average Reward across all environments, as the training progresses on POET with a 3 dimensional state space

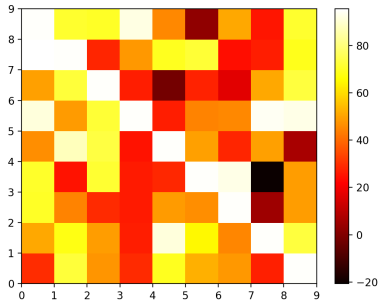

Figure 3: The cross evaluation on 9 environments. Values on the principal diagonal are close to the maximum obtainable reward indicating that agents trained on their corresponding environments perform well. Off diagonal values are slightly lower than principal diagonal indicating that the environments are tough, but the agents still achieve decent performance (Not negative). Finally, some agents achieve close to optimal rewards on environments they were not trained on indicating they might be goal switching pairs.

Many works dealing with coevolution in the open-ended paradigm rely on some kind of minimum criterion, where we do not want the environments to be too easy or too tough, judged by the agent's performance. These works rely on "novelty" or diversity in the environments generated to mitigate the above phenomenon. However as seen by this example, the environments might appear to differ enough, but still succumb to this loophole.

One safeguard to this can be to observe the similarity in the agent's behaviour or policy. With like-structured neural networks, this just means evaluating the similarity of two neural networks, for which many methods are available (Kornblith et al., 2019).

6.5 Object Embeddings, Limited Field of View

Our implementation, described in Section 6.3, had two major issues.

> **High-dimensional state space**: As we noted above, ES works well when the number of variables involved is small. The state space dimension directly affects the number of parameters of the policy network.

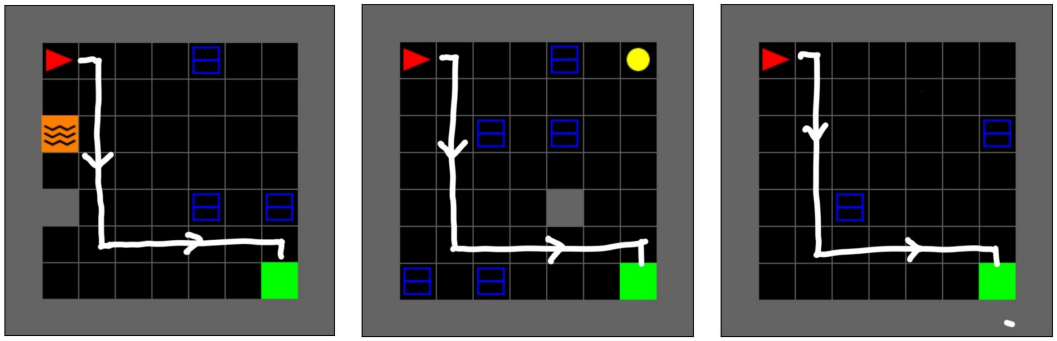

Figure 4: The agent learns the common path without obstacles to goal(shown in white) in the environments. Notice that the environments, decreasing in complexity from left to right, themselves are very different.

Object Encodings: These encodings can inadvertently induce ordinal relationships. For instance, if wall is 1, lava is 3, and ball is 4, then wall + lava = ball.

We proposed the following fixes:

Limited Field of View: Instead of the agent being able to view the entire grid, we only let it see it's immediate neighbours.

Object Embeddings: We use a matrix of size NUM_OBJECTS $\times$ EMBEDDING_DIM, initialized randomly, to "lookup" embeddings for each object. For instance, the embedding for wall is the 1^{st} row of this matrix.

6.6 Comparison with Direct Optimization

Figure 3 compares the median rewards obtained by direct optimization vs the agent output by POET as training progresses. To maintain a fair comparison, we account for the total number of episodes that POET agents have seen, and let direct optimization run for these many steps. We notice that the POET agent, that may have undergone goal switching outperforms direct optimization, hence validating the result from (Wang et al., 2019) on the minigrid environment.

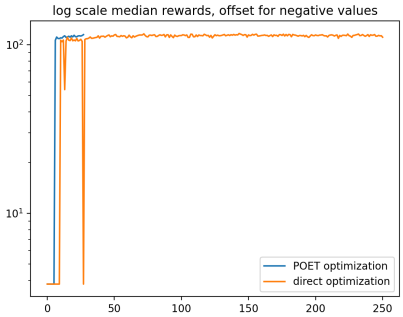

Figure 5: Comparison of Direct Optimization with POET on Median Rewards.

7 Conclusion

In this work, we applied the POET framework on the Minigrid Environment. To the best of our knowledge, this is the first application of POET to an environment other than the BiPedal Walker. We discussed the major changes that needed to be made in extending POET from BiPedal Walker to

other environments, with possibly discrete action spaces. In this, we showed that the evolutionary strategy used as the optimization procedure to train agents in POET is a bottleneck and requires hand-engineered, low-dimensional state representations. Changing the optimization strategy from ES to policy optimization methods like PPO, or TRPO can help overcome this limitation, and is left for future work. We showed how average reward can be decieving while monitoring an coevolution algorithm like POET, and proposed a cross evaluation metric that can give us better insights about the agents and environments. We further proposed a mutation strategy for the POET framework that favours reproducing environments with higher participation in goal switching. Finally, we also discussed how POET-like algorithms can succumb to shortcuts, and inadvertenly generate environments with similar solutions, and propose a way to mitigate them. We also open source our code at `https://github.com/ArjitJ/poet`

REFERENCES

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design, 2020.

Kuan Fang, Yuke Zhu, Silvio Savarese, and Fei-Fei Li. Adaptive procedural task generation for hard-exploration problems. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=8xLkv08d70T`.

gdicker1. gdicker1/gym-minigrid, 2021. URL `https://github.com/gdicker1/gym-minigrid`.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard Zemel, Wieland Brendel, Matthias Bethge, and Felix A. Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2(11):665–673, Nov 2020. ISSN 2522-5839. doi: 10.1038/s42256-020-00257-z. URL `http://dx.doi.org/10.1038/s42256-020-00257-z`.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited, 2019.

lcswillems. lcswillems/rl-starter-files, 2021. URL `https://github.com/lcswillems/rl-starter-files`.

Sebastien Racaniere, Andrew K. Lampinen, Adam Santoro, David P. Reichert, Vlad Firoiu, and Timothy P. Lillicrap. Automated curricula through setter-solver interactions, 2020.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions, 2019.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O. Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions, 2020.

A IMPLEMENTATION DETAILS

A.1 OBJECT ENCODINGS

OBJECT_TO_IDX = { 'unseen' : 0, 'empty' : 1, 'key' : 2, 'wall' : 3, 'floor' : 4, 'door' : 5, 'ball' : 6, 'box' : 7, 'goal' : 8, 'lava' : 9, 'agent' : 10, }

A.2 HYPERPARAMETERS

We describe the parameters used in our experiments. The initial learning rate was set to be 0.01 with a decay of 0.9999 and limit of 0.001. We use a training and evaluation batch size of 1. The noise standard deviation was set to 0.01 with a decay of 0.999 and limit of 0.01. We use a reproduction threshold of 10. The minimum criterion lower and upper threshold were 0 and 100 respectively. The obstacles we used were [lava, obstacle, box_to_ball, door, wall]. We run experiments with maximum number of environments set to 15. The number of steps before transfer was set to 25. We run the experiments for a maximum of 1500 iterations.

B ADDITIONAL EXPERIMENTS

B.1 SIGNIFICANCE OF REPRODUCTION THRESHOLD

If an environment has average reward greater than the reproduction threshold, then it is allowed to form child environments. We show here that the reproduction threshold is an important parameter for POET. In Figure 8 we show that a lower value of this threshold generates sub -ptimal env-agent pairs that result in negative average reward.

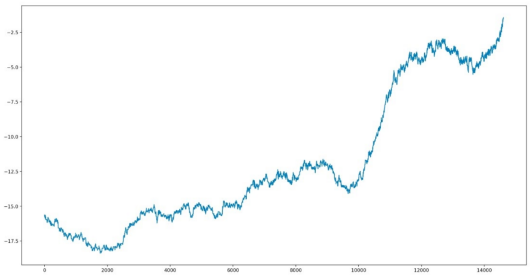

Figure 6: Reproduction threshold = -10

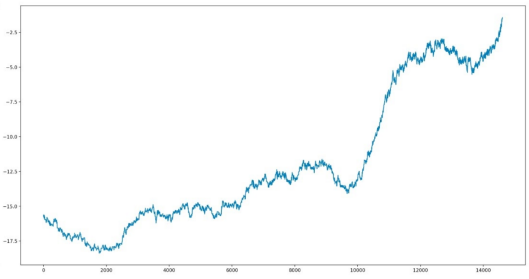

Figure 7: Reproduction threshold = 10

Figure 8: The plots show average reward vs episode number for different reproduction thresholds.