Scalable Evolutionary Hierarchical Reinforcement Learning

Sasha Abramowitz, Geoff Nitschke
abrsas002@myuct.ac.za,gnitschke@cs.uct.ac.za
University of Cape Town
South Africa

ABSTRACT

This paper investigates a novel method combining *Scalable Evolution Strategies* (S-ES) and *Hierarchical Reinforcement Learning* (HRL). S-ES, named for its excellent scalability, was popularised with demonstrated performance comparable to state-of-the-art policy gradient methods. However, S-ES has not been tested in conjunction with HRL methods, which empower temporal abstraction thus allowing agents to tackle more challenging problems. We introduce a novel method merging S-ES and HRL, which creates a highly scalable and efficient (compute time) algorithm. We demonstrate that the proposed method benefits from S-ES's scalability and indifference to delayed rewards. This results in our main contribution: significantly higher learning speed and competitive performance compared to gradient-based HRL methods, across a range of tasks.

CCS CONCEPTS

• **Computing methodologies** → **Evolutionary Strategies**; **Hierarchical Reinforcement Learning**.

KEYWORDS

Hierarchical Reinforcement Learning, Evolution Strategies

ACM Reference Format:
Sasha Abramowitz, Geoff Nitschke. 2022. Scalable Evolutionary Hierarchical Reinforcement Learning. In *Genetic and Evolutionary Computation Conference Companion (GECCO '22 Companion), July 9–13, 2022, Boston, MA, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3520304.3528937

1 INTRODUCTION

Reinforcement learning (RL) [18] has been used to create artificially intelligent agents for tasks ranging from robot locomotion [8] to board games such as chess and Go [17]. Many such agents use *Markov Decision Process* or *gradient*-based learning methods, such as *Deep Q-Networks* (DQNs) [11]. Single policy (flat) RL is generally used for relatively simple problems, however, increasingly complex problems (with sparse rewards or requiring multiple unrelated skills), are mostly unsolvable by current flat RL methods. To solve such RL problems (herein referred to as *hard* tasks) one can use *Hierarchical Reinforcement Learning* (HRL). HRL is a class of RL methods which excel at complex RL problems by decomposing them into sub-tasks, mimicking how humans build new skills on top of existing simpler skills. Gradient-based RL methods are also used by HRL and have solved various hard RL environments such as *Montezumas Revenge* [20] and generating complex robot behaviours [12, 20]. *Evolution strategies* (ES) [1] have also been applied to RL task environments, and have demonstrated competitive task-performance with flat gradient-based RL methods in robot locomotion and Atari game-playing [15]. However, ES has still not been applied to solve hard HRL problems.

ES has been used as a black-box optimizer for various tasks including, robot locomotion [3, 15] and loss function optimization [7]. There are many variants of ES [1], each with different evolutionary parameters. For example, CMA-ES [9] and $(1 + \gamma)$-*ES* [1], and *Scalable Evolution Strategies* (S-ES) [15].

All ES methods follow a scheme of *sample-and-evaluate* . Initially, policy variants are sampled around current policies parameters and variants are then evaluated to obtain fitness values. This provides information about the local fitness landscape, which is used to inform an update to the current policy. S-ES uses fitness to approximate a gradient and moves current policy parameters in a direction maximizing average reward. Given that ES is both a black-box process (making it indifferent to temporal details) and is a gradient-free method, it suffers from sub-optimal sample efficiency [16]. S-ES has demonstrated results comparable to gradient-based methods [15] on the *MuJoCo* and *Atari* [11] suite of benchmarks. However, S-ES has not been demonstrated on hard RL tasks (requiring long-term credit assignment), such as Montezuma's revenge [11] and robot locomotion with navigation [12].

HRL potentially solves much more complex tasks than flat RL methods, since HRL allows policies to abstract away large amounts of complexity and focus on solving simple sub-goals [12]. This is usually done by creating two classes of policies in a policy hierarchy: a *primitive* and a *controller*. The primitive is responsible for direct control of the agent and the controller manages the primitive, guiding its actions. For example, the HRL method *feudal-RL* [5] allows for communication between the controller and primitive by having the controller set goals for the primitive to complete.

Recent feudal-RL methods such as *FeUdal Networks for HRL* (FuN) [20] and *HRL with Off-Policy Correction* (HIRO) [12] have shown promise for learning sparse reward problems and hierarchies requiring complex primitives. HIRO uses a two-level hierarchy (one controller, one primitive) where the controller sets the goal and reward for the primitive. The goal takes various forms such as a position an agent must reach and reward is based on agent distance to the goal position. HIRO, FuN and related HRL algorithms use gradient-based RL methods to optimize their hierarchy of policies [12, 19, 20]. However, to date, non-gradient based RL solvers, such as ES, have not been extensively tested on hard RL problems (typically reserved for gradient-based HRL solvers).

ES has multiple advantages over gradient-based RL methods, but two of these advantages make ES especially suited for HRL

GECCO '22 Companion, July 9–13, 2022, Boston, MA, USA

ACM ISBN 978-1-4503-9268-6/22/07.
https://doi.org/10.1145/3520304.3528937

problems. First, it is invariant to delayed rewards and second, it has a more structured exploration mechanism [3, 15].

Furthermore, contrary to state-of-the-art RL and HRL methods, S-ES is highly robust to hyper-parameter changes [15]. Since HRL methods only introduce more hyper-parameters, the brittleness of current RL methods [8, 13] greatly increases HRL parameter tuning time. Thus, we introduce a new method[1] for training two-level policy hierarchies, optimized using a S-ES method: *Scalable Hierarchical Evolution Strategies* (SHES).

We compare SHES task-performance to other gradient-based HRL methods, also evaluated on the same tasks [12, 20]. The main objective is to demonstrate that SHES performs well on tasks that are challenging for gradient-based HRL methods and hence that S-ES is suitable for training hierarchies of policies. Our SHES method addresses various RL and HRL deficiencies by leveraging the benefits of S-ES to create an HRL method requiring minimal hyper-parameter tuning and that is competitive with state-of-the-art HRL methods across three hard HRL task environments.

2 METHODS

This section presents our method for learning hierarchical policies using S-ES: *Scalable Hierarchical Evolution Strategies* (SHES).

2.1 Policy Hierarchy

SHES is a Feudal RL [5] style method where high-level policies (controllers) direct and provide rewards to lower-level policies (primitives). The initial feudal RL method used a multi-level feudal hierarchy, whereas SHES uses a two-level hierarchy consisting of a single controller policy μ^c and a single primitive policy μ^p.

The controller sets goals and cannot directly perform task environment actions, while the primitive directly controls the agent with actions in the task-environment. Such actions aim to achieve goals set by the controller. More formally, given an environment state s_t, the controller produces a goal $g_t \in \mathbb{R}^d$ ($g_t \sim \mu^c(s_t)$). The controller produces g_t every c steps, where c is a hyper-parameter known as the *controller interval*. The goal is transformed using a static function such that it is always relative to the current state. For example, if the goal is a position it is updated every time-step so that the position is relative to the agent. The controller interval c is kept as a hyper-parameter since we observed that learning c often results in it degenerating into the simplest cases where c becomes 1 or the maximum episode length [20].

This provides the controller with a level of temporal abstraction which (for tested task-environments) enables it to plan a path without having to plan all agent actions required to follow this path. The primitive is passed the goal g_t and the state s_t and is tasked with reaching the goal. It samples an action $a_t \sim \mu^p(s_t, g_t)$ from its policy which is applied to the agent. The controller receives a reward from the environment, however is also responsible for rewarding its primitive. As in similar works, the primitive reward is based on its distance to its goal g_t [12, 20], however, the exact reward function is discussed in section 2.2.

In feudal RL, rewards are not shared between controller and primitive. For example, if the primitive reaches the goal set by the controller, but this does not provide a high enough reward then the primitive will receive a high reward, but the controller will not. This is known as *reward hiding* and is a key principle of feudal-RL [5]. SHES follows this primitive reward scheme and SHES does not share rewards between primitives and controllers [20], since it introduces an unnecessary hyper-parameter.

SHES is an extension of S-ES, where the main difference is that SHES co-evolves two policies and stores a set of parameters for both the controller θ^c and primitive θ^p. Every generation it creates n new controller and primitives pairs by perturbing the parameters θ^c and θ^p. Perturbation is done by adding a small amount of noise sampled from an n-variate Gaussian to the parameters $\theta^c_i = \theta^c + \epsilon^c \sim \mathcal{N}(0, \sigma^2)$. The primitive is similarly perturbed using a noise vector sampled from the same Gaussian $\epsilon^p \sim \mathcal{N}(0, \sigma^2)$. A shared noise table [15][2], allows for the sharing of common random numbers at negligible extra memory cost compared to single policy S-ES and increases the scalability and efficiency of SHES.

Each perturbed controller and primitive pair is evaluated in the task environment, where controller fitness is the cumulative environmental reward and primitive fitness is the cumulative reward from its controller. Both primitive and controller fitness are separately ranked and shaped as in S-ES. Ranked and shaped fitness is then used to approximate the gradients for the controller and primitive, which are optimized using the ADAM optimizer [10].

In feudal RL, controllers must adapt to non-stationary problems, since controllers and primitives learn simultaneously. That is, the controller learns not only how to solve the problem, but also how to recommend suitable goals to the current primitive. Such non-stationary problems are particularly challenging for many methods [12], however, SHES's robustness to noise made this trivial to solve. That is, the SHES controller simply interprets the primitive's changing behaviour as noise and suitably adapts its behaviour.

2.2 Primitive Reward

There are various ways to formulate the primitive reward [4, 12, 20], but in this study primitive reward is equated to the agent reaching its target consistently and quickly while avoiding local minima. SHES rewards the agent based on the portion of total distance covered, plus a bonus for reaching the target: $R^p_t = 1 - d_t/d_c + (1 if\, d_t < L\, else\, 0)$ Where, d_t: Euclidean distance between agent and goal, g_t: time-step t, d_c: distance at time-step, c: most recent time-step the controller recommended a goal, L: distance threshold ($L = 1$ here).

2.3 The Goal

A new goal is recommended once every c steps by the controller and for c-1 steps this goal is transformed using a fixed goal transition function. Each step, current goal g_t is concatenated onto the primitive's observations. We encode the primitive goal as the *sin* and *cos* of the angle from the agent to the goal, via allowing the controller to output a relative vector from the agent to the target and transforming this vector into an angle from agent to target. The *sin* and *cos* of this angle is the goal g_t passed to the primitive.

[1] https://github.com/sash-a/ScalableHrlEs.jl

[2] The sharing of common random numbers was shown by Salimans et al. [15] to allow for near linear speedup when scaling up to 1440 CPU cores.

2.4 Mutation Policy

The SHES method uses a *many-to-many* perturbation policy, meaning that each time a controller is perturbed a primitive is also perturbed. The benefit of this approach is that it decreases compute time and increases the sample efficiency since both the primitive and controller learn at the same time. However, it ranks controllers on an uneven playing field as they used different primitives variants, which can have a large impact on task performance.

2.5 Noise Sampling

SHES adapts S-ES antithetic sampling [2, 14], to operate on two policies and reduce variance. Since the controller and primitive both sample their own noise vectors (ϵ_c, ϵ_p), one way to perform antithetic sampling in SHES is to evaluate negatively perturbed policies ($-\epsilon_c, -\epsilon_p$) and positively perturbed policies ($+\epsilon_c, +\epsilon_p$).

Though, this leaves out two potential combinations when combining the positive with the negative perturbations. Using four perturbations instead of two led to a minor speed increase because of how it allows one to simplify the final matrix dot product when approximating the gradient, however it did not lead to a significant performance increase.

2.6 Speedup

Given that SHES is an extension of S-ES, it is expected to yield the same *linear speedup* and scalability benefits as S-ES [15]. Thus it is expected that SHES run-time is reduced by a factor of the number of cores used. The key difference between SHES and S-ES is communication overhead. However, the difference is minimal, since SHES communicates an extra three numbers (one 32 bit float and two integers), for each evaluation to represent the performance of two policies instead of one. Given the small amount of extra data sent between CPU cluster nodes, we do not expect the extra communication of SHES to significantly impact its speedup.

3 EXPERIMENTS

SHES was evaluated, in comparison to HIRO, in three environments (*Ant Gather*, *Ant Maze*, *Ant Push*). These tasks were selected since each is defined by sparse rewards or deceptive local minima and require learning both robot locomotion and navigation behaviour.

Ant Gather: The agent receives +1 reward for each *food* object it collects and −1 reward for each *poison* object it collects (figure 1, left). Food and poison are placed in random positions at the start of each episode. The evaluation score is defined as the maximum reward throughout an episode.

Ant Maze: The agent must learn to reach a target within a U-shaped maze. During training the target is randomly generated, however at evaluation time it is placed on the opposite side of the maze (red dot in figure 1, centre). Evaluation reward is based on the distance to the target and at evaluation time the agent is given a score of 1 if it is within 5 units of the final step of the episode.

Ant Push: The agent must learn to push the red block to the right allowing it to reach the end goal at the top of the maze (red dot in figure 1, right). It is rewarded based on its distance to the red dot and receives an evaluation score of 1 if within 5 units of the red dot on the final step of the episode.

SHES experiments were executed on a cluster comprising Intel Xeon 24 core CPUs running at 2.6GHz with 64GB of RAM. HIRO was executed on the same hardware with a Nvidia V100 GPU.

4 DISCUSSION

Figures 2 (a), (b) and (c) indicate SHES is competitive with HIRO [12]. In *Ant Gather* SHES outperforms the maximum evaluation reward yielded by HIRO by a factor of 1.24, and significantly exceeds the mean evaluation reward of our own HIRO experiments ($p = 0.008$, Mann–Whitney U test [6]). For *Ant Push* (Figure 2, c), SHES evaluation reward significantly exceeds our HIRO method re-run ($p = 0.0007$) and SHES on higher core counts (600) outperforms the original HIRO experiments by achieving the maximum evaluation reward of 1. However, for *Ant Maze* (Figure 2, b), results indicate that SHES is unable to learn to solve the task (thus flat evaluation reward results), and significantly under-performs compared to our HIRO method ($p < 0.0001$).

These results demonstrate that SHES outperforms HIRO on two of the three tasks. However, this comes at the cost of sample efficiency. SHES requires over 100× more samples than HIRO to attain this performance. Though, this is not unexpected given that gradient-free optimization has been shown to be less sample efficient than gradient-based optimization [16].

SHES does, however, out-perform HIRO in terms of run-time, when given sufficient compute. For example, for the *Ant Gather* task (figure 2, a), SHES task-performance over time, rises faster than HIRO indicating increased learning speed, as well as demonstrating the distributed computing benefits of SHES. In the case of the *Ant Push* task (Figure 2, c), SHES yields a comparable task-performance to HIRO, however, later task-performance (after approximately 3 hours) rises faster than HIRO further supporting the learning speed and distributed computing benefits of SHES.

Also consider, SHES (600 cores) matched the evaluation score of HIRO (Nachum et al.) in under an hour, on both the *Ant Gather* and *Ant Push* tasks. Replication of HIRO on these tasks (10 million training steps [12]), took over 12 hours to achieve the same evaluation score when executed on an Nvidia V100 GPU. Thus SHES offers at least a 12× compute speedup and increased task performance over the HIRO method. Also, the high task-performance of SHES on *Ant Gather* (figure 2, a), further supports the SHES method's indifference to delayed rewards in HRL problems. *Ant gather* is the task environment (of tasks tested) with the most sparse reward and is the task on which SHES yields the highest task-performance relative to HIRO, supporting SHES applicability to sparse reward problems and its benefit over gradient-based methods.

5 CONCLUSION

The main contribution of this work was a new evolutionary HRL method: SHES. Across all hard HRL tasks tested, SHES achieved a high learning speed, as well as significantly out-performing a current state of the art off-policy HRL method (HIRO) in most environments tested. In comparison to HIRO, SHES performed especially well on sparse reward RL tasks. SHES addresses a need for computationally expedient HRL methods that yield high-task performance across a range of HRL (and more generally RL) tasks.

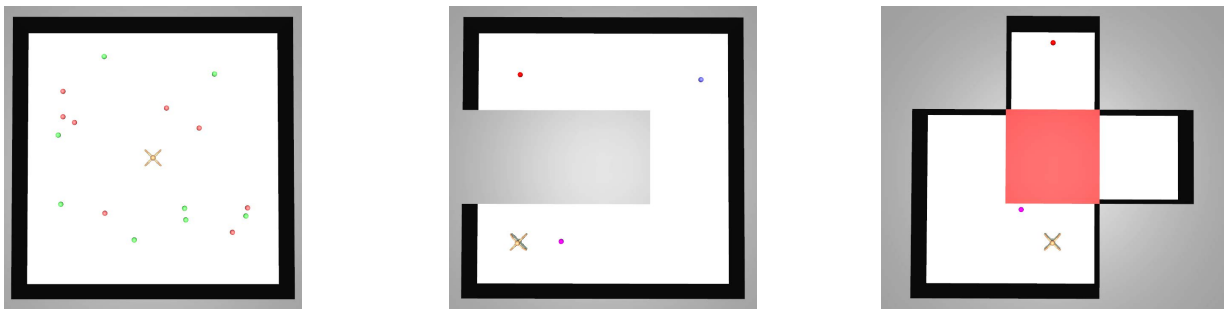

Figure 1: *Ant Gather* environment (left) *Ant Maze* environment (centre) and *Ant Push* environment (right). These three task-environments were used to evaluate the SHES method.

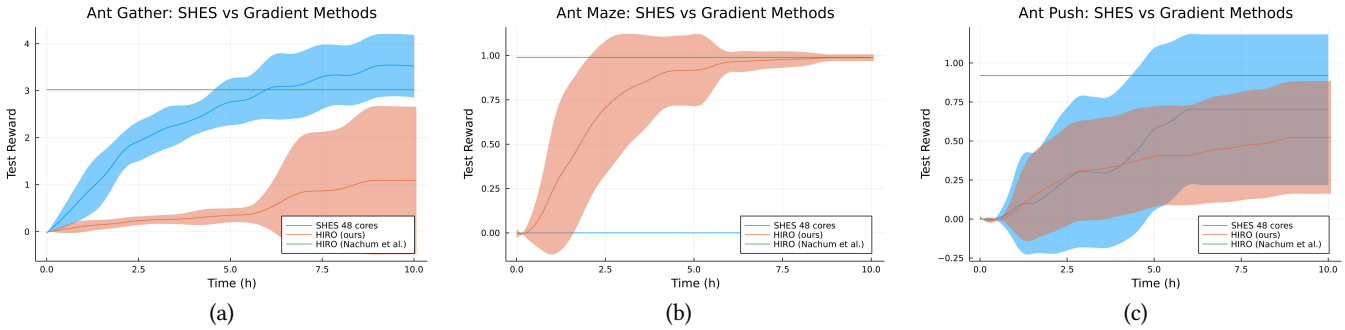

Figure 2: SHES, HIRO (48 CPU cores) average evaluation reward (10 runs) in (a): *Ant Gather*, (b): *Ant Maze*, (c): *Ant Push*. HIRO Nachum et al. maximum evaluation reward (not reward over time) is also plotted. Shaded regions show standard deviation.

REFERENCES

[1] Hans-Georg Beyer and Hans-Paul Schwefel. 2002. Evolution strategies – A comprehensive introduction. *Natural Computing* 1 (2002), 3–52. Issue 1. https://doi.org/10.1023/A:1015059928466

[2] Dimo Brockhoff, Anne Auger, Nikolaus Hansen, Dirk V Arnold, and Tim Hohm. 2010. Mirrored sampling and sequential selection for evolution strategies. In *International Conference on Parallel Problem Solving from Nature*. Springer, 11–21.

[3] Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. 2017. Improving Exploration in Evolution Strategies for Deep Reinforcement Learning via a Population of Novelty-Seeking Agents. *Advances in Neural Information Processing Systems* 2018-December (12 2017), 5027–5038.

[4] Erwin Coumans et al. 2013. Bullet physics library. *Open source: bulletphysics. org* 15, 49 (2013), 5.

[5] Peter Dayan and Geoffrey E Hinton. 1993. Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, S. Hanson, J. Cowan, and C. Giles (Eds.), Vol. 5. Morgan-Kaufmann.

[6] B. Flannery, S. Teukolsky, and W. Vetterling. 1986. *Numerical Recipes*. Cambridge University Press, Cambridge, UK.

[7] Santiago Gonzalez and Risto Miikkulainen. 2020. Improved training speed, accuracy, and data utilization through loss function optimization. In *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–8.

[8] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*. PMLR, 1861–1870.

[9] Nikolaus Hansen and Andreas Ostermeier. 2001. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation* 9, 2 (2001), 159–195.

[10] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[12] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. 2018. Data-Efficient Hierarchical Reinforcement Learning. *Advances in Neural Information Processing Systems* 31 (2018), 3303–3313.

[13] Tom Le Paine, Cosmin Paduraru, Andrea Michi, Caglar Gulcehre, Konrad Zolna, Alexander Novikov, Ziyu Wang, and Nando de Freitas. 2020. Hyperparameter selection for offline reinforcement learning. *arXiv preprint arXiv:2007.09055* (2020).

[14] Hongyu Ren, Shengjia Zhao, and Stefano Ermon. 2019. Adaptive Antithetic Sampling for Variance Reduction. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 5420–5428. https://proceedings.mlr.press/v97/ren19b.html

[15] Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *CoRR* abs/1703.03864 (2017). arXiv:1703.03864 http://arxiv.org/abs/1703.03864

[16] Olivier Sigaud and Freek Stulp. 2019. Policy search in continuous action domains: an overview. *Neural Networks* 113 (2019), 28–40.

[17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

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

[19] Richard S Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence* 112, 1-2 (1999), 181–211.

[20] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. 2017. Feudal networks for hierarchical reinforcement learning. In *International Conference on Machine Learning*. PMLR, 3540–3549.