Towards Run-time Efficient Hierarchical Reinforcement Learning

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

***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.**

***Index Terms*—Hierarchical Reinforcement Learning, Evolution Strategies, Ant Gather, Ant Maze, Ant Push**

I. Introduction

Reinforcement learning (RL) [1] has been used to create artificially intelligent agents for tasks ranging from robot locomotion [2] to video games such as *StarCraft* [3] and board games such as Chess and Go [4]. Many such agents use *Markov Decision Process* or *gradient* based learning methods, such as *Deep Q-Networks* (DQNs) [5] and policy gradient methods [1], [6]. 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 excelling 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 *Montezuma's revenge* [7], [8] and generating complex robot behaviours [7], [9]. *Evolutionary strategies* (ES) [10] 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 [11]. 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, optimizing designs in structural and mechanical engineering problems [13], robot locomotion [11], [14], [15] and loss function optimization [16]. There many variants of ES [10], each with different evolutionary parameters. For example, CMA-ES [17] and $(1+\gamma)$-ES [10], and *Scalable Evolution Strategies* (S-ES) [11].

All ES methods use a *sample-and-evaluate* scheme (figure 1). 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 [18]. However, Liu et al. [19] showed promising results addressing this inefficiency using trust regions which allowed for monotonic improvement [19]. S-ES has demonstrated results comparable to gradient-based methods [11], on benchmark tasks including *MuJoCo* [20] and *Atari* game playing [5]. However, S-ES has not been demonstrated on hard RL tasks (requiring long-term credit assignment), such as *Montezuma's revenge* [5] and robot locomotion [9], [21].

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 [9], [21]. 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* [22], 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) [7] and *HRL with Off-Policy Correction* (HIRO) [9] have shown promise for learning sparse reward problems and hierarchies requiring complex primitives. For example, HIRO uses a two-level hierarchy (one controller and one primitive) where the controller sets the goal and reward for the primitive. The goal can take various forms such as a position an agent must reach in the task environment and the reward is based on agent's distance to the goal position.

978-1-6654-6708-7/22/$31.00 ©2022 IEEE

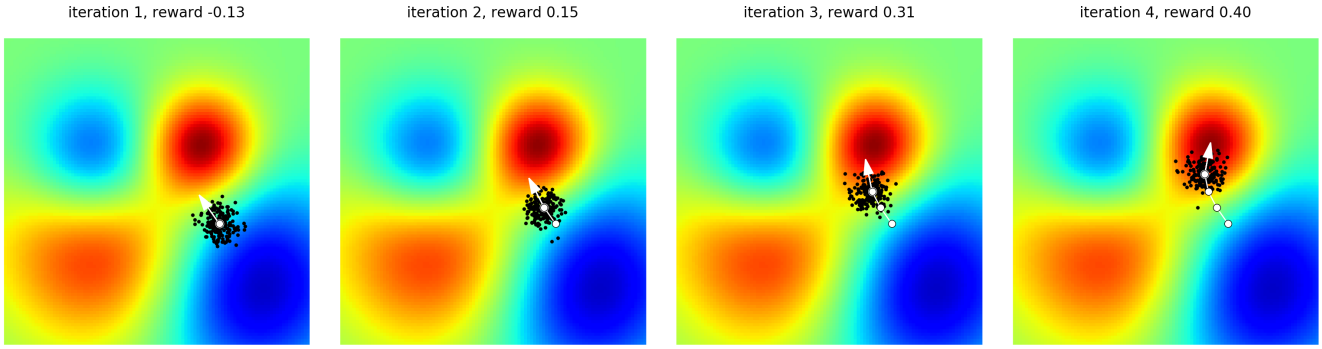


Fig. 1. ES optimization example in a 2D problem space. The white dot is the main policy and the black dots are its perturbations. The main policy is guided from areas of low reward (blue) to areas of high reward (red). Image from: Karpathy et al. [12].

HIRO, FuN and related HRL algorithms use gradient-based RL methods to optimize their hierarchy of policies [7]–[9], [23]. 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).

Another option to adapt RL agents to solve hard tasks is *transfer learning*. This entails training an agent to solve some simple task and then transferring the learnt policy (behaviour) to a related hard task. This allows the agent to get a *warm start* in the hard task by using policy information gained from the related easier task. There are various approaches to transfer knowledge to an RL agent to boost task-performance on hard tasks [24], [25], such approaches can be combined with HRL policies to improve task-performance or sample efficiency of HRL methods [9], [21]. ES has many advantages over gradient based RL methods, but two advantages make ES especially suited for HRL problems. First, it is invariant to delayed rewards and second, it has a more structured exploration mechanism [11], [14].

Robustness to delayed rewards is especially useful for hard HRL tasks defined by long-term credit assignment. Similarly, many hard RL problems are also defined by multiple large local minima, requiring effective exploration methods to solve. Such advantages suggest that ES, specifically S-ES, will out-perform gradient based HRL solvers across various hard HRL problems. Furthermore, contrary to state-of-the-art RL and HRL methods, S-ES is highly robust to hyper-parameter changes [11]. Since HRL methods only introduce more hyper-parameters, the brittleness of current RL methods [2], [26] 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 [7], [9], [21], [27]. 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.

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

II. Methods

This section presents our method for learning hierarchical reinforcement learning policies using evolutionary strategies: *Scalable Hierarchical Evolution Strategies* (SHES).

A. Policy Hierarchy

SHES is a *Feudal* RL [22] style method where high level policies (controllers) direct and provide rewards to lower level policies (primitives). The initial feudal RL method [22] 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. The aim of such actions is 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 as 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 [7].

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

Algorithm 1 SHES Algorithm

Input: Learning rate α, noise standard deviation σ, roll-outs n, initial policy parameters θ^c and θ^p

for t = 0,1,2... **do**
  for i = 1,2,...,n **do**
    Sample $\epsilon_i^c, \epsilon_i^p \sim \mathcal{N}(0, I)$ $f_i^c, f_i^p = F(\theta_t^c + \epsilon_i^c * \sigma, \theta_t^p + \epsilon_i^p * \sigma)$
    $\theta_{t+1}^c = \theta_t^c + \alpha \frac{1}{n\sigma} \sum_{i=1}^n f_i^c \epsilon_i^c$
    $\theta_{t+1}^p = \theta_t^p + \alpha \frac{1}{n\sigma} \sum_{i=1}^n f_i^p \epsilon_i^p$
  end for
end for

to follow this path. The primitive is passed the goal g_t and the state s_t and 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 [7], [9], however the exact reward function is discussed in section II-B.

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 [22]. SHES follows this primitive reward scheme and SHES does not share rewards between primitives and controllers [7], 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 $\theta_i^c = \theta^c + \epsilon^c \sim \mathcal{N}(0, \sigma^2)$. 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 [11][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 learning speed 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 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 [28]. 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 [9], 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]The sharing of common random numbers was shown by Salimans et al. [11] to allow for near linear speedup when scaling up to 1440 CPU cores.

B. Primitive Reward

There are various ways to formulate the primitive reward [7], [9], [29], 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 (given a recommended position), plus a bonus for reaching the target (Equation 1).

$$R_t^p = 1 - d_t/d_c + (1 \; if \; d_t < L \; else \; 0) \tag{1}$$

Where, d_t is the Euclidean distance between the agent and the goal g_t at time-step t, d_c is the distance at time-step, c (most recent time-step controller recommended a goal), and L is a distance threshold ($L = 1$ in this study). This improves upon simply rewarding the primitive with the negative distance by allowing it to be positive if the primitive performs well thus avoiding a large local minima and normalizing the distance to make it agnostic to target distance. Also, adding an extra reward for being close to the target incentivises the agent to reach the goal as quickly as possible to maximize the time for which it receives this extra reward.

This was found to be the best performing primitive reward given the deficiencies observed for primitive rewards used by HIRO [9] and FuN [7]. HIRO rewarded primitives with negative distance to the goal g_t [9], encouraging agents to move to the target quickly. However this introduced local minima where the agent could simply *die* instantly thus avoiding anymore negative reward. FuN rewarded its primitive based on the *cosine* similarity of the path the agent took since the goal was suggested and the straight line from the agent's position to the goal. This encouraged the primitive to follow a specific path, making it more predictable for the controller, but this reward put little emphasis on speed of learning.

C. The Goal

A new goal is recommended every c steps by the controller and for the next c-1 steps this goal is transformed using a fixed goal transition function. Each step the current goal g_t is concatenated onto the primitive's observations. The primitive goal g_t is the vector from agent position to the goal recommended by the controller. This goal recommendation was selected given the difficulty for primitives to learn suitable representations, observed in related approaches. For example, in HIRO [9], the goal passed to the agent is the entire state space, so the primitive must attempt to match the position of all agent joints and the overall position of the agent. This HIRO [9] approach limits types of usable

primitive rewards and increases learning difficulty for the primitive, hence our selected goal recommendation approach.

As with primitive rewards, goal encoding has a significant impact on task-performance. An obvious goal encoding to use a vector from the agent's position to goal g_t. We found that this did not work since the values are not normalized, thus the primitive ANN performs worse because of non-normalized input data [30]. However, normalizing the goal vector means the agent no longer has any notion of distance to goal g_t. We solve this by concatenating the distance to the goal onto a normalized vector, where the distance is scaled down to an appropriate range (dividing it by 1000). We encode the primitive goal as the sin and cos of the angle from the agent to the goal g_t. This was done by allowing the controller to output a relative vector from the agent to the target and transforming this vector into an angle from the agent to the target. The sin and cos of this angle is the goal g_t passed to the primitive.

D. Transfer Learning

Transfer learning presents another option for learning hierarchies of policies and as such will serve a suitable comparison to SHES. There are various transfer learning methods for RL and HRL [24], [25], [31], but we use a simple pre-trained primitive [32], which is pre-trained using SHES. We call this method SHES-TL. Pre-training is only for the primitive as one cannot pre-train a controller without already having a trained primitive. Thus training is split into two phases, first the primitive pre-training and second combined training where the controller makes use of the pre-trained primitive. Pre-training allows all controllers from each task environment to be tested using the same primitive, thus improving sample efficiency since the primitive can be trained once for many similar tasks.

First, the SHES primitive is trained using a random controller, thus promoting the generality of the primitive. Once the primitive has achieved the desired performance it is saved and used as the primitive for controllers during training on hierarchical environments. Next, hierarchical training has two options for running the pre-trained primitive, its weights can be unfrozen or frozen meaning that it either continues to learn with the controller or it keeps its task-performance from the pre-training stage. SHES-TL uses a frozen primitive since preliminary experiments demonstrated that frozen primitives significantly out-performed unfrozen primitives.

Thus, given a frozen primitive, SHES-TL benefits from a general and reusable primitive, mitigating the non-stationary problem-space problem for controller adaptation. However, SHES-TL requires more samples as the primitive and controller do not use the same samples (as is the case in SHES). A limitation of pre-training is the primitive observation space during pre-training must match its observation space during hierarchical training. Since the observation spaces of task-environments used in our experiments differ, one cannot train a primitive with the exact same observations as required in each of the environments. As such we extract common observations for all the environments and create a pre-training environment using these observations. This requires the primitive's observations in the hierarchical environment are sliced so the primitive only receives the observations it was pre-trained with. This mandates manual selection of observations and gives transfer learning an advantage as the primitive only receives the observations it needs for learning. Given this advantage and transfer learning mitigating the non-stationary problem, SHES-TL is a suitable comparative method for SHES.

E. 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. This is presented in algorithm 1 (line 5), where F evaluates a perturbed controller $\theta_t^c + \epsilon_i^c * \sigma$ and a perturbed primitive $\theta_t^p + \epsilon_i^p * \sigma$. 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.

Another option is a *one-to-many* perturbation policy. This approach alternates between only perturbing the primitive while using the main controller, or only perturbing the controller while using the main primitive. This alleviates conceptual drawbacks by allowing controller variants to be ranked on equal grounds because they all use the same primitive. However, it was found that this approach had a large impact on sample efficiency, thus the *many-to-many* approach is used.

F. Noise Sampling

SHES adapts S-ES antithetic *mirrored sampling* [33], [34], so as 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 the pair of 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 led to a minor speed increase because of how it allows one to simplify the final matrix dot product when approximating the gradient. That is, perturbing four times instead of two, simplifies the dot product even further, since there are four fitness values using the same noise vector. For large dot products this greatly reduces the computation time by reducing the length of each matrix by a factor of four. Hence, SHES performs four perturbations instead of two.

G. 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 [11]. 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 in order 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 computational speedup.

III. Experiments

SHES was evaluated versus HIRO and SHES-TL (section II) in the *Ant Gather*, *Ant Maze*, *Ant Push* tasks (left, center, and right in figure 2, respectively). These tasks were selected since each is defined by sparse rewards and require learning both robot locomotion and navigation behavior.

Ant Gather: The agent receives $+1$ reward for each green *food* object it collects and -1 reward for each red *poison* object it collects. Food and poison are generated in random positions each time the environment is initialized. Test score is defined as the maximum reward throughout an episode.

Ant Maze: The agent must learn to move to the opposite side of the maze (red dot in figure 2, center). During training the agent must reach a randomly generated target inside the maze (blue dot in figure 2, center), and is rewarded based on the distance to this target. During testing the agent must come sufficiently close to the red dot by the final episode iteration to receive a test score of 1, otherwise the agent receives 0.

Ant Push: The agent must learn to push the red block to right allowing it to reach the end goal at the top of the maze (red dot in figure 2, right). It is rewarded based on its distance to the red dot and receives a test score of 1 if within 5 units of the red dot by the final episode iteration.

All task environment simulations were re-implemented in Julia[3](using the same assets and physics engine as previous work [9]), and SHES, SHES-TL experiments were executed on a cluster (2, 10, 25 nodes) comprising Intel Xeon 24 core CPUs running at 2.6GHz with 64GB of RAM. HIRO was executed on an Intel Xeon 24 core CPU running at 2.6GHz with 32GB RAM and an Nvidia V100 GPU. All experiments were run 10 times. Results graphs (figure 3, 4 and 5) present the mean value (*test reward* and *training reward*) with standard error. Figure 3 presents test reward, whereas figures 4 and 5 presents training reward. All experiments were run for a maximum time of 10 hours and a maximum of 3000 generations, and experiments halted when one stopping condition was reached. For example, in the case of SHES, a higher the core count resulted in more generations completed in 10 hours and a sufficiently high core count resulted in the generation limit being reached before 10 hours (figure 4).

Comparative method results for HIRO, are taken from previous work [9]. This method was selected since it is a HRL method that currently yields state-of-the-art task-performance across the *Ant Gather*, *Ant Maze* and *Ant Push* tasks [9]. We also re-ran the HIRO[4] [9] method (10 runs) on all tasks, in order to gauge mean task-performance, compared to SHES.

IV. Results and Discussion

Figures 3 (a), (b) and (c) indicate SHES is competitive with HIRO [9]. In *Ant Gather* SHES out-performs the maximum test reward yielded by HIRO by a factor of 1.24, and significantly exceeds the mean test reward of our own HIRO experiments ($p = 0.008$, Mann–Whitney U test [35]). For *Ant Push* (Figure 3, c), SHES test reward significantly exceeds our HIRO method re-implementation and re-run ($p = 0.0007$). However, for *Ant Maze* (Figure 3, b), results indicate that SHES is unable to learn to solve the task (thus flat test reward results), and significantly under-performs compared to our HIRO method ($p < 0.0001$). These results demonstrate that SHES out-performs HIRO on two of the three tasks. However, this comes at the cost of sample efficiency. SHES requires over $100\times$ more samples than HIRO to attain this performance. Though, this is not unexpected given that gradient free optimization has been demonstrated as less efficient than gradient based optimization [18].

For example, Salimans et al. [11] presented S-ES sample efficiency up to approximately 8 times worse than *Trust Region Policy Optimization* (TRPO) [36] on simple 2D locomotion environments. Thus, one expects even more sample inefficiency on harder 3D locomotion and navigation tasks, meaning the sample inefficiency of SHES is countered by the fact that HIRO (as an off-policy HRL method), was specifically designed to be sample efficient. SHES does however, out-perform HIRO in terms of run-time, when given sufficient compute. For example, for the *Ant Gather* task (Figure 3, a), SHES task-performance over time, rises faster than HIRO indicating increased learning efficiency, as well as demonstrating distributed computing benefits of SHES. That is, as presented in figure 4, for all tasks, learning efficiency of SHES (600 cores) exceeds SHES (240 cores), which in turn exceeds SHES (48 cores). In the case of the *Ant Push* task (Figure 3, c), SHES yields a comparable task-performance to HIRO, however, later task-performance (after approximately 3 hours) rises faster than HIRO further supporting learning speed and distributed computing benefits of SHES (figure 4).

Also, the high task-performance of SHES on *Ant Gather* (Figure 3, a), further supports the benefit of the SHES method's indifference to delayed rewards in HRL problems. That is, *Ant Gather* is the task environment (of the three tasks tested) with the most sparse reward and is the task on which SHES yields the highest task-performance relative to HIRO. This supports the applicability of SHES to sparse

[3]https://github.com/sash-a/HrlMuJoCoEnvs.jl

[4]https://github.com/tensorflow/models/tree/master/research/efficient-hrl

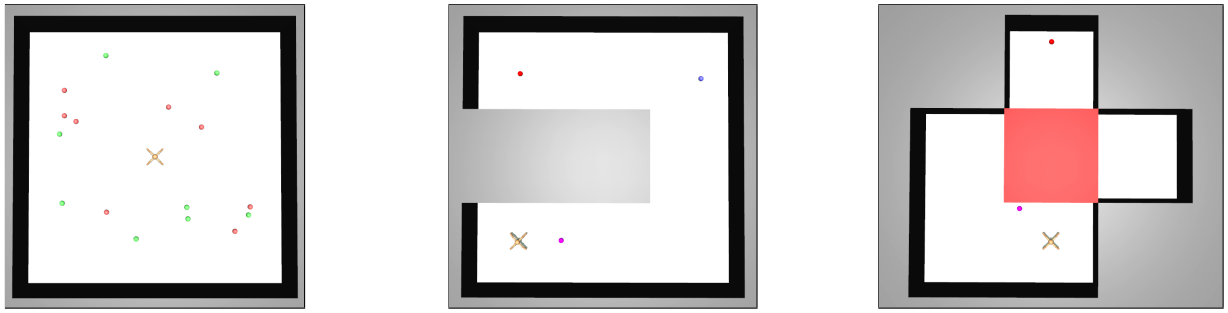
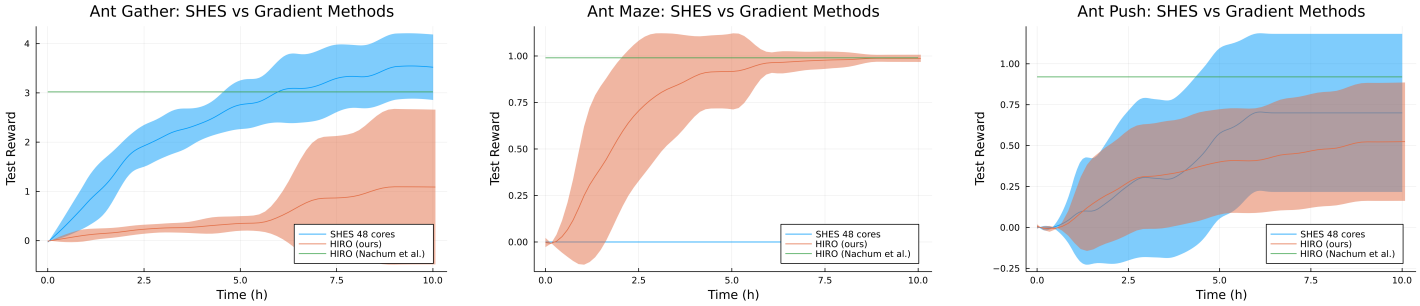
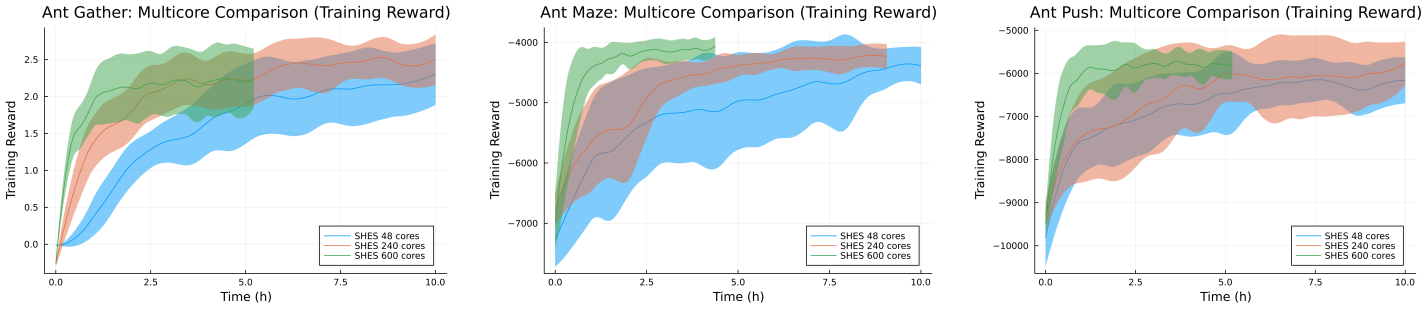


Fig. 2. *Ant Gather* (left) *Ant Maze* (center) and *Ant Push* environments (right). These three task-environments were used to evaluate the SHES method.



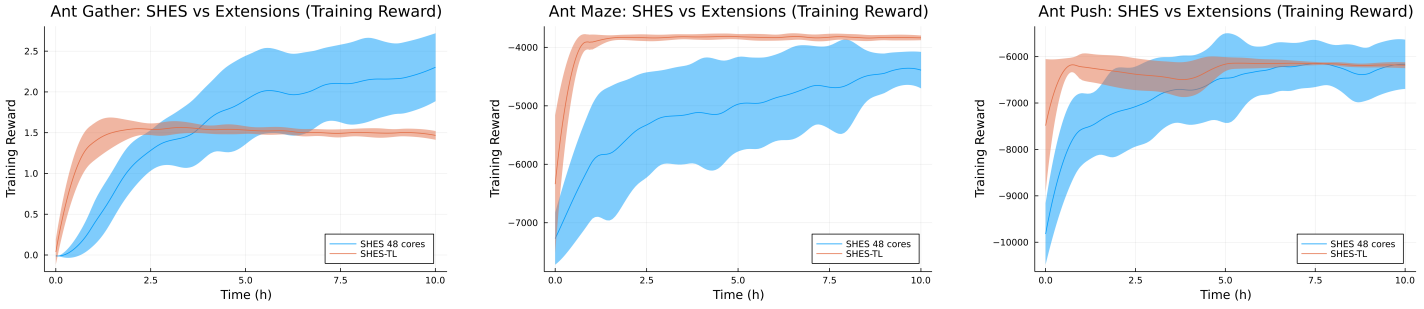
(a) (b) (c)

Fig. 3. SHES versus HIRO (48 CPU cores) average task-performance (10 runs) in (a): *Ant Gather*, (b): *Ant Maze*, (c): *Ant Push*. Colored regions show standard deviation. HIRO (Nachum et al.) maximum test reward (not reward over time) is also plotted.



(a) (b) (c)

Fig. 4. Training reward, showing impact of (48, 240, 600) CPU cores in (a): *Ant Gather*, (b): *Ant Maze*, (c): *Ant Push*.



(a) (b) (c)

Fig. 5. Training reward (showing transfer learning impact) of SHES, SHES-TL in (a): *Ant Gather*, (b): *Ant Maze*, (c): *Ant Push*).

Hyper-parameter	SHES, SHES-TL	HIRO
Controller Interval	25	10
Controller distance	4	10
Learning rate	0.01	0.001
Sigma	0.02	*n/a*
Episodes per policy	5	*n/a*
Policy per generation	1000	*n/a*
Time horizon (Agent lifetime)	500	500
Experiment parameter	**SHES, SHES-TL**	**HIRO**
Runs	10	10
Generations	3000	3000
Run-time Limit	10 hrs	10 hrs
CPU Cores	48, 240, 600	24 + Nvidia V100 GPU

TABLE I
EXPERIMENT AND METHOD (SHES, HIRO) PARAMETERS

reward problems and its benefit over gradient based methods.

The relatively poor performance of SHES on the *Ant Maze* task (Figure 3, b), versus its significantly higher task performance on the *Ant Gather* and *Ant Push* tasks (Figure a, c), compared to HIRO is hypothesized to be a result of the short time horizon that the agent is given (500 time-steps), since when testing with longer time horizons (1000 time-steps), SHES was able to easily solve the *Ant Maze* task. This is supported by previous work [37]. This is further support by observing agent behavior in the maze environment[5]. Observing the best performing SHES policies in the *Ant Maze* task, it is notable that the agent reaches most targets, but targets sufficiently far across the maze are unreachable given the short time horizon. Thus, SHES agents only learn suitable solutions when agent lifetime is sufficiently long to complete the task.

Figure 5 presents comparative results (mean training reward) of SHES versus SHES-TL on the *Ant Gather*, *Ant Maze* and *Ant Push* tasks. For all environments SHES-TL improves learning efficiency (speed) as evidenced by the steeper rise of the SHES-TL curves for each task (figure 5 a, b, c). However, these results do not include the time taken to pre-train the primitive, though the same primitive was used for all tasks and could be used for multiple other tasks, thus reducing pre-training time. These results demonstrate the benefits of SHES-TL in the form of learning efficiency and task performance, but also show that SHES yields comparable task performance compared to a transfer learning method in two of the three tasks tested (figure 5 a, c).

Figure 6 presents the SHES method's computational speedup as a function of number of cores plotted versus perfect linear speedup. SHES has sub-linear speedup, but with a linear trend. This is not unexpected given the strictly serial parts of SHES, namely fitness shaping, ranking and the gradient calculation. SHES offers approximately a one sixth speedup for each core added, which was tested up to 600 cores. Interestingly SHES is able to achieve more speedup per core than S-ES, this is likely due to the higher proportion of parallel work it needs to perform, because of its extra policy. This computational speedup as a function of increasing numbers of cores is one of the main benefits of SHES. To further illustrate, consider that SHES (600 cores) was able to match the test score of HIRO in under an hour, on both the *Ant Gather* and *Ant Push* tasks. Replication of HIRO on these tasks (running for 10 million training steps [9]), took over 12 hours to achieve the same test score when executed on an Nvidia V100 GPU. Thus SHES offers at least a $12\times$ compute speedup and increased task performance over HIRO.

[5]https://github.com/sash-a/ScalableHrlEs.jl/blob/master/README.md

V. CONCLUSIONS

The main contribution of this work was a new evolutionary HRL method: SHES. Across most hard HRL tasks tested, SHES achieved significant computational speedup and increased learning efficiency as well as out-performing a current state of the art off-policy HRL method: HIRO. In comparison to HIRO, SHES performed especially well on sparse reward RL tasks and also performed comparably to pre-trained variants of the same method. Thus, the SHES method addresses a current need for computationally expedient HRL methods that yield high-task performance across a range of HRL (and more generally RL) tasks.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[3] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

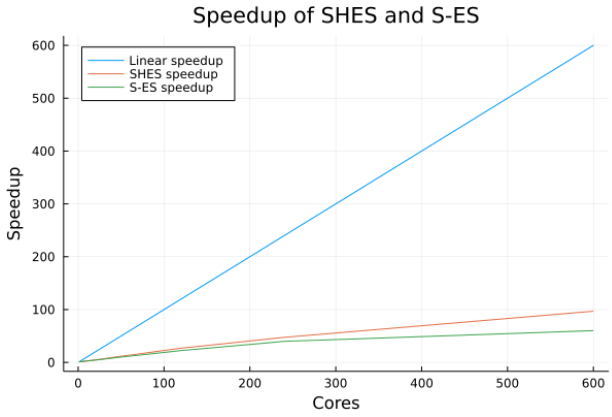


Fig. 6. SHES and S-ES speedup versus perfect linear speedup: SHES offers approximately a one fifth speedup for each core added up to 250 cores and one sixth speedup up to 600 cores, notably this outperforms S-ES.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour *et al.*, "Policy gradient methods for reinforcement learning with function approximation." in *NIPs*, vol. 99. Citeseer, 1999, pp. 1057–1063.

[7] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2017, pp. 3540–3549.

[8] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," in *International Conference on Machine Learning*. PMLR, 2020, pp. 507–517.

[9] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 31, pp. 3303–3313, 2018.

[10] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies – a comprehensive introduction," *Natural Computing*, vol. 1, pp. 3–52, 2002.

[11] T. Salimans, J. Ho, X. Chen, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," *CoRR*, vol. abs/1703.03864, 2017.

[12] A. Karpathy, T. Salimans, J. Ho, P. Chen, I. Sutskever, J. Schulman, G. Brockman, and S. Sidor, "Evolution strategies as a scalable alternative to reinforcement learning," Mar 2017.

[13] S. Datoussaïd, G. Guerlement, and T. Descamps, "Applications of evolutionary strategies in optimal design of mechanical systems," *Foundation of Civil and Environmental Engineering*, vol. 7, pp. 35–51, 2006.

[14] E. Conti, V. Madhavan, F. P. Such, J. Lehman, K. O. Stanley, and J. Clune, "Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents," *Advances in Neural Information Processing Systems*, pp. 5027–5038, 12 2017.

[15] A. Katona, D. W. Franks, and J. A. Walker, "Quality evolvability es: Evolving individuals with a distribution of well performing and diverse offspring," *arXiv preprint arXiv:2103.10790*, 2021.

[16] S. Gonzalez and R. Miikkulainen, "Improved training speed, accuracy, and data utilization through loss function optimization," in *2020 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2020, pp. 1–8.

[17] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary computation*, vol. 9, no. 2, pp. 159–195, 2001.

[18] O. Sigaud and F. Stulp, "Policy search in continuous action domains: an overview," *Neural Networks*, vol. 113, pp. 28–40, 2019.

[19] G. Liu, L. Zhao, F. Yang, J. Bian, T. Qin, N. Yu, and T. Y. Liu, "Trust region evolution strategies," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33. AAAI Press, 7 2019, pp. 4352–4359.

[20] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 5026–5033.

[21] C. Florensa, Y. Duan, and P. Abbeel, "Stochastic neural networks for hierarchical reinforcement learning," *arXiv preprint:1704.03012*, 2017.

[22] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Advances in Neural Information Processing Systems*, S. Hanson, J. Cowan, and C. Giles, Eds., vol. 5. Morgan-Kaufmann, 1993.

[23] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[24] Z. Zhu, K. Lin, and J. Zhou, "Transfer learning in deep reinforcement learning: A survey," *arXiv preprint arXiv:2009.07888*, 2020.

[25] M. Hawasly and S. Ramamoorthy, "Lifelong transfer learning with an option hierarchy," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 1341–1346.

[26] T. L. Paine, C. Paduraru, A. Michi, C. Gulcehre, K. Zolna, A. Novikov, Z. Wang, and N. de Freitas, "Hyperparameter selection for offline reinforcement learning," *arXiv preprint arXiv:2007.09055*, 2020.

[27] R. Houthooft, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel, "Vime: Variational information maximizing exploration," *Advances in Neural Information Processing Systems*, vol. 29, pp. 1109–1117, 2016.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[29] E. Coumans *et al.*, "Bullet physics library," *Open source: bulletphysics. org*, vol. 15, no. 49, p. 5, 2013.

[30] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Transactions on nuclear science*, vol. 44, no. 3, pp. 1464–1468, 1997.

[31] S. Schaal, "Is imitation learning the route to humanoid robots?" *Trends in cognitive sciences*, vol. 3, no. 6, pp. 233–242, 1999.

[32] B. Zoph, G. Ghiasi, T.-Y. Lin, Y. Cui, H. Liu, E. D. Cubuk, and Q. Le, "Rethinking pre-training and self-training," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds. Curran, Inc., 2020, pp. 3833–3845.

[33] H. Ren, S. Zhao, and S. Ermon, "Adaptive antithetic sampling for variance reduction," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. PMLR, 09–15 Jun 2019, pp. 5420–5428.

[34] D. Brockhoff, A. Auger, N. Hansen, D. V. Arnold, and T. Hohm, "Mirrored sampling and sequential selection for evolution strategies," in *International Conference on Parallel Problem Solving from Nature*. Springer, 2010, pp. 11–21.

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

[36] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*. PMLR, 2015, pp. 1889–1897.

[37] R. Wang, S. S. Du, L. Yang, and S. Kakade, "Is long horizon RL more difficult than short horizon RL?" in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds. Curran, Inc., 2020, pp. 9075–9085.