

Robust Reinforcement Learning via Genetic Curriculum

Yeeho Song¹ and Jeff Schneider¹

Abstract—Achieving robust performance is crucial when applying deep reinforcement learning (RL) in safety critical systems. Some of the state of the art approaches try to address the problem with adversarial agents, but these agents often require expert supervision to fine tune and prevent the adversary from becoming too challenging to the trainee agent. While other approaches involve automatically adjusting environment setups during training, they have been limited to simple environments where low-dimensional encodings can be used. Inspired by these approaches, we propose genetic curriculum, an algorithm that automatically identifies scenarios in which the agent currently fails and generates an associated curriculum to help the agent learn to solve the scenarios and acquire more robust behaviors. As a non-parametric optimizer, our approach uses a raw, non-fixed encoding of scenarios, reducing the need for expert supervision and allowing our algorithm to adapt to the changing performance of the agent. Our empirical studies show improvement in robustness over the existing state of the art algorithms, providing training curricula that result in agents being 2 - 8x times less likely to fail without sacrificing cumulative reward. We include an ablation study and share insights on why our algorithm outperforms prior approaches.

I. INTRODUCTION

When training an RL agent, learning to solve the remaining 10% of the scenarios is often significantly more difficult compared to learning to solve the first 90% of the scenarios. This presents a challenge to using RL in safety critical applications, such as autonomous vehicles, where robustness, the probability of an agent not landing in irreversible and catastrophic states (i.e. collision) plays a crucial role in determining product viability. In a typical RL setup, as an agent’s performance improves, it becomes not only rare to encounter and collect data on the scenarios in which the agent does poorly but also difficult to learn new behaviors when approaching a local minimum. This results in an agent converging to a suboptimum with several scenarios left unsolved.

One prominent approach for robust RL is to use adversarial agents to inject adversarial noise to explore challenging situations. However, adversarial agents often converge to the worst case scenario in which the protagonist cannot learn and requires expert supervision to avoid this issue. Some scenarios are not well represented by adversarial noise, such as a particular sequence of tasks or environment setup difficult for the agent. While other approaches involve encoding the environment or generating a curriculum to help learn difficult tasks, they are mostly limited to benchmarks with small scenario space where low-dimensional encodings can be used.

In this paper, we propose genetic curriculum (GC) which uses a genetic algorithm to generate curricula for training robust RL agents. By running a genetic algorithm, GC will generate training scenarios that the agent cannot solve, helping the agent to explore the scenario space efficiently. As scenarios generated by the genetic algorithm will be similar to each other, a skill learned from one scenario can easily be transferred to another scenario, allowing them to work as a curriculum helping the agent to learn faster and converge to a more optimal policy. As our algorithm is non-parametric, it can use raw scenario encoding of non-fixed length, minimizing expert supervision of designing encoding methods and helping support highly complex scenario description as the agent’s performance improves.

II. RELATED WORKS

In robust RL where an agent should be trained against and verified in a variety of different scenarios, recent advances in sim2real [1]–[4] and high fidelity simulators [5], [6] makes it feasible to collect realistic training data in scenarios too dangerous and difficult to collect in real life. However, even with this setup, an agent would often leave a long tail of unsolved scenarios. As an agent becomes more robust, it becomes less likely to encounter and collect data from situations where the agent fails. Even when data is available, it is often difficult to learn new skills as the agent would often be approaching a local minimum optimized towards more probable scenarios.

Adversarial training is one such method for gathering data in the region where the agent does not do well. Showing success with classical RL [7]–[9] and deep learning architectures [10]–[13], adversarial training in RL pairs a protagonist agent with an adversary agent each playing a zero-sum game of maximizing/minimizing reward in the environment. Robust adversarial RL (RARL) [14] uses an adversary to apply external force to the protagonist. Risk averse robust adversarial RL (RARARL) [15], probabilistic action robust Markov decision process (MDP) and robust action robust MDP (PRMDP / NRMDP) [16] uses adversaries to inject action noise. However, some challenging situations are difficult to represent as noise, such as particularly hard scenarios or environment setups. Also, such a min-max setup often leads to the adversary quickly converging to a worst-case scenario too difficult for the protagonist to learn. Our approach differs by not only encoding scenarios and environment setup instead of adversarial noise but also generating supporting scenarios that help the agent to learn new skills.

Fingerprint policy optimization (FPO) [17] shares insights on encoding environments and scenarios. Building upon

¹Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, USA. {yeehos, jeff4, }@andrew.cmu.edu
This work was partly funded by The Boeing Company,

the previous works on classical RL [18] [19], FPO uses Bayesian optimization to select the training setup with the biggest expected performance improvement. However, such approaches have been limited to low-dimensional fixed-length encoding for training environments. Our approach differs by using non-parametric optimizers to use non-fixed length encoding. This allows us to minimize expert supervision by directly using the raw values of a simulator while being more versatile to adapt to the agent’s changing needs with no information loss.

Curricular learning explores generating supporting scenarios to help learn new skills. Organizing training data to gradually introduce more complex concepts, [20], curricular learning has shown success in supervised learning tasks [21]–[25] as well as various RL tasks [26]–[33]. Automatically generating a curriculum of similar yet gradually more complex scenarios is an ongoing question in curriculum learning. Self-paced deep RL (SPDL) [34] is one such approach of exploring how curriculum can automatically be generated based on the agent’s current performance. At each epoch, SPDL locates the distribution of scenarios the agent currently performs well and will select training scenarios as a distribution progressively moving towards the goal distribution. However, SPDL likewise has been limited to low-dimensional fixed length encoding of scenarios. We will explore non-parametric approaches to expand the ideas to non-fixed raw encoding of curriculum.

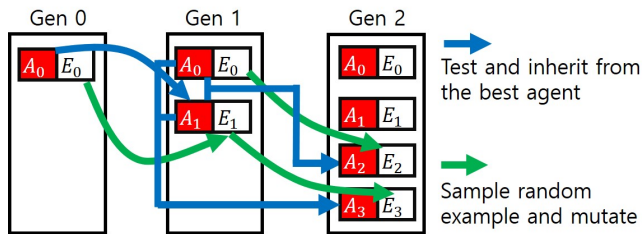


Fig. 1. Curriculum generation during POET [35]. Unlike our proposed approach, POET runs training inside genetic algorithm, greatly increasing computational load.

Paired open-ended trailblazer (POET) [35] borrows several elements from genetic algorithms to use non-fixed length encodings. As shown in Figure 1, POET starts by adding a pair consisting of a random agent (A_0) and a random training example (E_0). During an epoch, all pairs that haven’t reached a satisfactory performance are trained in parallel. At the end of each epoch, a new pair is generated by adding a small random perturbation, or mutation, to an existing example and pairing it up with a new agent that inherits the parameter weights of the existing agent that performs best on the new example. As POET has training nested inside the genetic algorithm, the algorithm is computationally expensive as resources spent by any agents not contributing to the performance of the best-performing agent are wasted. Our approach is computationally more efficient by separately running the genetic algorithm outside of the loop. At the end of each epoch, the current policy is fixed and the genetic algorithm runs to generate a set of scenarios the agent cannot solve. Furthermore, POET uses mutation to generate new scenarios. This makes the search

for new scenarios an inefficient random walk and makes it difficult to advance towards scenarios drastically different from the starting scenario. Our approach on the other hand incorporates crossover, the processing of mixing sequences from two parent sequences to generate two offspring, to help cover a wider variety of scenarios faster.

III. BACKGROUND

This paper examines continuous space MDP represented as a tuple: $[S, A, P_\psi, r_\psi, \gamma]$ where S is a set of states, A action space, and $\gamma \in [0, 1)$ is the temporal discount factor. Scenario ψ are not fully observable to the agent, such as obstacles situated out of the line of sight or an internal systems failure. P_ψ and r_ψ represent the state transition dynamics and reward function of the scenario. In the event of a partial engine failure related to fuel pumps, the engine would be burning less fuel per second and will be delivering less thrust, hence the state transition probability and reward on fuel usage will be different from those of a nominal scenario. The agent’s policy, $\pi(a | s)$ maps states $s \in S$ to $a \in A$. The utility of a policy π for scenario ψ is the expected return, $J_\psi(\pi) = \mathbb{E}_{a_t \sim \pi} \sum_t \gamma^t r_\psi(s_t, a_t)$.

During training, an RL algorithm seeks the optimal policy π^* by exploring and gathering data about reward and state dynamics. The data gathered will be dependent on the distribution of scenarios the agent experienced during training $p_{train}(\psi)$;

$$\pi^* = \underset{\psi}{argmax} \sum_{\psi} p_{train}(\psi) J_\psi(\pi) \quad (1)$$

IV. PROBLEM STATEMENT

For a given scenario, ψ , we measure success as follows;

$$G_\psi(\pi) = \begin{cases} 0, & \text{if } \pi \text{ fails for } \psi \\ 1, & \text{otherwise} \end{cases} \quad (2)$$

Failure is defined as failing to achieve a goal before exhausting a resource budget set by the user. This could be a walking robot falling down before reaching a target, a wheeled robot crashing into a stationary object, a manipulator robot reaching certain time steps with a cumulative reward lower than a threshold. A robust algorithm should minimize the probability of failure during testing;

$$\pi_{robust}^* = \underset{\psi}{argmax} \sum_{\psi} p_{test}(\psi) G_\psi(\pi) \quad (3)$$

Ideally, the trained agent should satisfy the robustness criteria $\pi^* = \pi_{robust}^*$. As the distribution of scenarios encountered during testing, $p_{test}(\psi)$, and the definition of failure $G_\psi(\pi)$, are problem specific, most papers focus on $J_\psi(\pi)$ and $p_{train}(\psi)$. While reward shaping with $J_\psi(\pi)$ is possible, such as giving a high penalty towards failure, this often requires expert supervision and fine-tuning for the training to be stable. We, therefore, take the curricular approach of investigating how $p_{train}(\psi)$ can be better selected to train a robust agent.

V. APPROACH

To train a robust agent, we select training scenarios as scenarios the agent currently fails in. Solving these examples directly addresses Equation (3). We also select the scenarios to be similar to each other. This follows the idea of curricular learning on building a set of similar scenarios with varying types of challenges and levels of difficulty to help transfer skills from one task to another more easily. Also, just as adversarial RL adds perturbations to make an agent robust to a variety of situations, the differences in our scenarios will help an agent learn not only a specific task but also a variety of similar tasks as well. This paper proposes GC, which borrows concepts from genetic algorithms and curriculum learning to achieve these goals.

At each epoch, curriculum generation starts by initializing a population $\Psi_{population}$ of size M_{pop} with randomly generated scenarios. We express scenarios as a sequence of non-fixed length $\psi = (z_0, z_1, z_2, \dots)$ where each vector z defines the order of values to be used which would otherwise be filled in by a random number generator in the original benchmark. Factors of variation include size and duration of obstacles and bumps on terrain to the time of occurrence, or type and magnitude of an actuator failure of a legged robot depending on the benchmark. At each iteration, $\Psi_{population}$ is evaluated by current policy π . ψ is appended to $\Psi_{training}$ if π is unable to solve ψ .

The next $\Psi_{population}$ is generated by crossover. With $L(\psi)$ as the length of encoding for ψ , the probability of a scenario being chosen as a parent for a crossover operation is higher if the scenario’s encoding is shorter. This encourages sequences to only retain the sections critical to failure and avoid having offspring diverse in irrelevant ways. A selected parent change a random section of its encoding with a random section of another parent’s encoding as shown in Figure 2. The crossover operation repeats until $|\Psi_{population}| \geq M_{pop}$. To introduce new vectors to the gene pool, every ψ in $\Psi_{population}$ has mutation probability p_{μ} . The mutation is equivalent to conducting crossover with a randomly generated sequence as shown in Figure 2.

Once $|\Psi_{population}| \geq M_{pop}$, GC exits the scenario generation cycle and $\Psi_{training}$ is used to train π for an epoch. Algorithm 1 shows the pseudocode of our proposed approach.

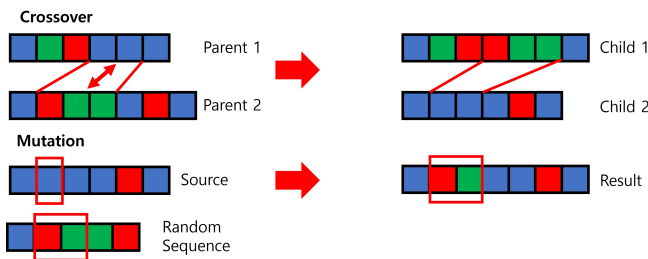


Fig. 2. Visualization on crossover and mutation. Our approach use genetic algorithm to use raw, non-fixed length encoding to generate similar scenarios that can act as a curriculum and dynamically change length of encoding to keep up with agent’s performance.

$$p_{parent}(\psi) = \begin{cases} \frac{1}{(\max(L(i)) - L(i) + 1)}, & \text{if } \pi \text{ fails } \psi \\ 0, & \text{otherwise} \end{cases} \quad (4)$$

Algorithm 1 Genetic Curriculum (GC)

```

1: Initialize Policy  $\pi_0$ 
2: Input training steps, iterations, epochs,  $M_{train}$ ,  $M_{pop}$ ,  $p_{\mu}$ 
3: for  $i$  in epochs do
4:   Initialize  $\Psi_{population}$ 
5:   Initialize  $\Psi_{train} = \{\}$ 
6:   for  $k$  in iterations do
7:     Fitness = evaluate( $\Psi_{population}, \pi_{i+1}$ )
8:      $\Psi_{train} = \text{collect}(\Psi_{train}, \Psi_{population}, \text{Fitness})$ 
9:     if  $|\Psi_{train}| > M_{train}$  then
10:       Break
11:     end if
12:      $\Psi_{population} = \text{crossover}(\Psi_{population}, M_{pop}, \text{Fitness})$ 
13:      $\Psi_{population} = \text{mutate}(\Psi_{population}, p_{\mu})$ 
14:   end for
15:   while steps < training steps do
16:      $\pi_{i+1} = \text{Train Agent}(\pi_i, \Psi_{train})$ 
17:   end while
18: end for

```

Our algorithm has several desirable features. As a non-parametric optimizer, it is easy to adapt to various types of tasks and policies. ψ not only has no fixed length, but as visualized in Figure 2, offspring scenarios can easily get longer or shorter during curriculum generation. This allows encoding length to expand and contract as needed to accommodate changes in the agent’s performance over time. During the experiments, the encoding dimension dynamically changed from 20 - 300D, which would have been difficult with Bayesian optimization as used in FPO [17]. Another feature visible in Figure 2 is that scenarios within a curriculum will be similar to each other. With the crossover and mutation operations, all scenarios have part of their sequence shared recurring in another scenario. This similarity makes it easier to transfer skills from one to another.

VI. EXPERIMENTS

A. Benchmarks

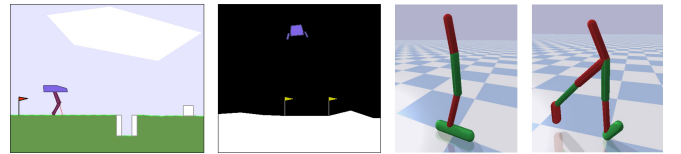


Fig. 3. Screenshot of Benchmarks used in this paper (from left to right), BipedalWalker(Hardcore/System), LunarLander, Hopper, and Walker which tests agent’s robustness against a variety of obstacle courses / actuator failure.

BipedalWalkerHardcore [36] involves agent observing the world with LIDAR, IMU, and joint encoder values to control torque on each of the bipedal walker’s leg servos. The goal is to traverse through a randomly generated obstacle course filled with stairs, pitfalls, and walls. While individual obstacles are easy, the challenge is to learn a robust policy that can solve a variety of sequences of obstacles without falling.

BipedalWalkerSystem is a modified version of the above where the agent traverses through a fixed sequence of obstacles with simulated random system failures. When a failure is triggered at a random timestep, the affected servo will only be able to deliver 60 -100% of the original power

depending on the severity. While individual scenarios are easy, the challenge is to learn a robust kinetic energy management skill to go through obstacles even if an actuator fails.

LunarLander is a modified benchmark of the one provided by [36]. Using position and velocity observations, an agent has to safely land on the landing pad using its main engine (ME) and side thrusters. When failure is triggered at a random timestep, the throttle of the affected rocket motor is limited to 60 - 100% of the nominal power. Under nominal scenarios, the best policy is to wait until the last moment and fire the ME at full thrust to minimize fuel used. However, if a system failure occurs, the lander may crash due to ME being unable to provide enough thrust. A robust policy should keep the rate of descent to a manageable level to maximize the possibility of landing even when a failure occurs.

Hopper and **Walker** are modified benchmarks based on the original versions provided by [37]. When a random system failure occurs, a torque limit of 75 - 100% of the nominal maximum is applied to the affected servo. To make the benchmarks more challenging, we mount a simulated payload of sizes 0.75 and 0.5 on the Hopper and Walker legged robots. A policy is considered to have failed if its simulated payload touches the ground.

B. Baseline Algorithms

Some of the state of the art approaches have been chosen as follows. To compare GC against adversarial RL approaches, we chose RARL [14], RARARL [15], and PRMDP / NRMDP [16]. We chose FPO [17] for comparison against approaches that control the training environment. For comparison against curricular RL, we chose SPDL [34] for parametric curricular approaches and POET [35] for non-parametric approaches.

The algorithms in this paper require a base RL algorithm for updating policies. RL algorithms listed on top of the respective leaderboards for the original version of the benchmarks are used as base RL algorithms. BipedalWalker and LunarLander use soft actor-critic (SAC) [38], while Hopper and Walker use twin delayed DDPG (TD3) [39]. SAC uses $\gamma = 0.99$, learning rate of $1e-4$, batch size of 100, and replay memory size of $1e6$, while TD3 uses $\gamma = 0.98$, learning rate of $3e-4$, batch size of 100, and replay memory size of $2e5$. Both algorithms use fully connected networks consisting of layers sized 400 and 300 updated by ADAM [40] and activated with the ReLU function.

C. Evaluation and Hyperparameters

One of the main challenges for comparing the performance of each algorithm is the vastly different computation requirements of each algorithm during training. FPO, POET, SPDL, and ours require additional steps for evaluating the agent’s performance. While this can be expensive, evaluation can not only run in parallel but is also cheaper than exploration which requires backpropagation. Adversarial RL algorithms, on the other hand, had extra computational costs for training both protagonist and adversarial networks at the same time. As we are concerned about how robust a converged solution

TABLE I
TRAINING DURATION AND TESTING RESOLUTION

Benchmark	Steps per Epoch	Testing Set Size	Number of Epochs
BipedalWalkerHardcore	$1e5$	1000	350
BipedalWalkersystem	$1e5$	2500	30
LunarLander	$1e4$	2500	80
Walker	$5e4$	2500	60
Hopper	$5e4$	2500	40

TABLE II
HYPERPARAMETER SEARCH ON BASELINE ALGORITHMS

Algorithm	Parameter	Tested	Selected
RARAL	α	0.05,0.1,0.5	0.1
RARARL	ξ	1,5,10,20	10
PRMDP	α	0.05,0.1,0.3,0.5	0.1
NRMDP	α	0.05,0.1,0.3,0.5	0.05

is, we report results based on how many epochs have passed. Each epoch consists of the same numbers of policy updates and exploration steps per benchmark. To share insights in cases where the total number of environment interactions is more important, we also include a separate set of experiments on the LunarLander benchmark where values are reported based on how many steps each algorithm interacted with the simulator.

We report performance with mean and standard error on 10 random seeds per algorithm per benchmark, with testing sets consisting of randomly generated scenarios. For BipedalWalker benchmarks, only 3 random seeds were used to balance accuracy and computational cost. This is because it would take 7 - 14 days to approach convergence for such benchmarks. In the case of POET where multiple agents are trained simultaneously, we report the performance of the best agent in terms of reward as the result of the random seed. Table I shows the length of each epoch, testing set size, and the number of epochs for each benchmark.

To offer a fair comparison, a hyperparameter search is conducted for adversarial RL algorithms as shown in Table II. Hyperparameters that performed the best overall throughout the benchmarks were selected.

The size of policy evaluation for FPO, POET, SPDL is the same as the size of policy evaluation used for reporting performance during training. POET also requires manual reward thresholds on what is considered as not too trivial nor difficult before adding an scenario for training. BipedalWalker benchmarks use the same threshold of 50 - 300 as used in the original POET paper. For other benchmarks, we selected the value by checking their training curves and marking when the reward starts to climb and flatten. The threshold is set as 100 - 250 for LunarLander and 1000 - 2000 for Walker and Hopper benchmarks.

The default versions of the benchmarks random use number generators to create scenarios at each run. For FPO and SPDL, we engineered a fixed-length encoder where the range of the numbers coming out of the random number generator was defined. For POET and our method, the string of numbers to be used in the place of the random number generator was stored in a sequence.

For GC, the curriculum size is 300 for BipedalWalker

benchmarks and 100 for the rest, which is a rounded value on how many times the simulators are reset per each epoch. The size of parent and offspring populations is 100 each which is a rounded value on the minimum size required to have at least two or three failure sequences upon random initialization to act as parents for subsequent generations. While hyperparameter tuning was also conducted on p_{μ} , GC didn't show much sensitivity towards p_{μ} and a value of 0.1 is used. The GC's reliance on crossover more than the mutation rate is highlighted in the ablation study.

D. Ablation Study

To better understand how our approach help improves the robustness of an agent, an ablation study with BipedalWalkerHardcore is conducted. When generating a curriculum filled with failure scenarios, NoMutation turns off mutation, NoCrossover turns off crossover, and RandomFailure fills a curriculum with examples that are randomly generated and are unsolvable when tested by current policy. To see how a genetic algorithm can generate a curriculum of similar examples and its effect on performance, the mean genetic distance of a curriculum is also reported. Every time a new example is loaded during training, genetic distance is calculated by counting the minimum number of variables that have to be changed, added, and deleted to convert the previous example to the new example. Single Run provides additional data on the effect of genetic distance on robustness by generating a curriculum consisting of one failure scenario, making the mean genetic distance of the curriculum to be zero.

VII. RESULTS

A. Comparison with Baseline Algorithms

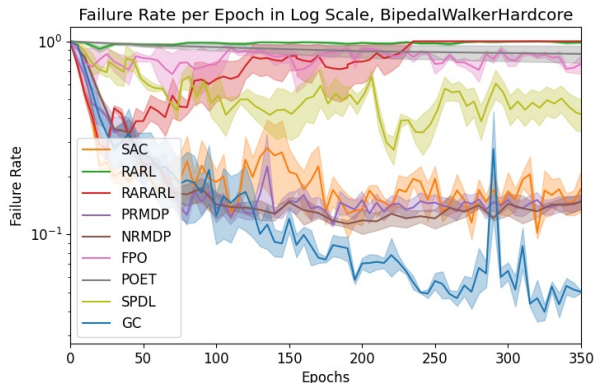


Fig. 4. Training Curve for BipedalWalkerHardcore. Lower the better

As shown in Figure 4 and Table III, our proposed GC consistently improves over the state of the art algorithms especially in terms of robustness. Agents trained by GC are 2 - 8x times less likely to fail compared to those trained on baseline SAC / TD3.

One interesting observation from Table III is that even when agents show quite a difference in performance in terms of robustness, such differences are less obvious when looking at rewards only. When trained, big reward coming from the majority of the cases tends to dominate over bad

rewards coming from the minority of cases. During training, similar issues are observed where the failure rate continues to converge when the reward does not show significant progress. This highlights the challenge of capturing robustness alone by reward and using it to optimize the policy.

The results from FPO and SPDL highlight the GC's benefit of using raw scenario encoding of non-fixed length. As a non-parametric optimizer, GC can adapt the length of its encoding to generate more complex scenarios as the protagonist agent's performance improves. The effect is most well observed in the case of BipedalWalkerHardcore where GC expanded the encoding size by 10 - 15 times during training to precisely describe the what training scenarios should be.

Also, the results from adversarial RL show selecting a supporting curriculum is as important as generating challenging scenarios during training. While adversarial agents can generate challenging situations during training, they do not present in a way that the trainee can easily learn new skills. The challenges of injecting difficult cases without supporting a curriculum are further highlighted in the following section on the ablation study.

The effect of not nesting training within the genetic algorithm can be observed from POET's results. The original POET paper reported a reward of around 250 when trained and tested on scenarios based on the BipedalWalkerHardcore. POET showed a similar performance when evaluated on the training set during our experiments. However, when evaluated against the testing set which includes the entire scenario space as designed by the benchmark, POET performed poorly agent's skills were not generalizable across a wide variety of scenarios. As POET is computationally expensive and relies only on mutation for curriculum generation, each trained agent could only experience less training data from a less diverse set of scenarios compared to those from GC.

An interesting observation from the trained policies is while there are some scenarios where an agent had more difficulty solving than the others, there was no clear trend describing which scenario is objectively more difficult than others or a priori difficult scenarios where solving one case means being able to solve all the easier cases. When a scenario that a trained agent consistently fails are used as a training scenario to a randomly initialized agent, the agent would learn how to solve the scenario. However, regardless of the random seed used, finding a general policy that solves all the scenarios was difficult. This shares insight that a robust training scheme should not only focus on performance per each task but also on learning a general skillset applicable across the tasks.

B. Comparison with Respect to Environment Interactions

One of the important criteria in RL is how efficiently it can learn per the number of environment interactions. Figure 5 shows that while GC is a bit slow at the start due to the extra cost of running genetic algorithms, the cost is offset by having better training examples. Unlike the baseline RL (SAC) and adversarial RL methods where marginal utility per environment interaction quickly diminishes, GC can sustain

TABLE III
REWARD AND MEAN FAILURE RATE OF TRAINED AGENTS(%)

Reward					
Algorithm	BipedalWalkerHardcore	BipedalWalkerSystem	LunarLander	Walker	Hopper
Base RL (SAC / TD3)	291.76 ± 17.41	300.94 ± 1.85	265.30 ± 1.92	2300.81 ± 29.30	2266.64 ± 3.05
RARL	7.67 ± 13.49	289.25 ± 5.08	28.00 ± 12.24	122.60 ± 4.58	203.12 ± 4.53
RARARL	230.14 ± 19.52	270.89 ± 13.59	272.29 ± 0.80	2156.48 ± 10.31	2199.82 ± 3.18
PRMDP	285.30 ± 25.66	298.42 ± 0.23	260.73 ± 2.28	2165.19 ± 11.36	2275.72 ± 2.04
NRMDP	289.82 ± 19.25	291.42 ± 5.05	254.99 ± 1.27	2147.51 ± 1.30	2092.10 ± 3.85
FPO	118.60 ± 1.21	286.47 ± 10.84	256.31 ± 4.61	2134.83 ± 5.09	2044.73 ± 3.21
POET	24.60 ± 18.61	-62.58 ± 14.14	213.30 ± 3.94	2068.50 ± 31.01	2129.93 ± 1.81
SPDL	305.90 ± 0.45	289.13 ± 5.19	221.31 ± 10.71	589.78 ± 74.04	2274.70 ± 13.06
GC (Proposed)	304.33 ± 1.65	300.00 ± 1.00	272.82 ± 0.30	2342.61 ± 5.45	2283.48 ± 2.01
Failure Rate(%)					
Algorithm	BipedalWalkerHardcore	BipedalWalkerSystem	LunarLander	Walker	Hopper
Base RL (SAC / TD3)	10.20 ± 0.71	3.62 ± 0.58	5.19 ± 1.20	4.31 ± 1.00	12.99 ± 3.52
RARL	91.27 ± 3.28	5.97 ± 2.87	73.56 ± 13.52	79.01 ± 16.82	85.61 ± 9.77
RARARL	27.69 ± 3.14	15.29 ± 5.27	4.33 ± 0.88	3.85 ± 0.41	14.36 ± 5.27
PRMDP	11.23 ± 0.36	2.85 ± 0.40	2.24 ± 0.90	3.88 ± 1.75	12.46 ± 4.70
NRMDP	11.00 ± 1.27	5.02 ± 1.38	6.41 ± 1.15	6.96 ± 2.09	28.32 ± 5.91
FPO	67.60 ± 19.05	8.30 ± 4.55	11.73 ± 2.71	12.12 ± 4.83	38.33 ± 5.11
POET	84.96 ± 9.45	100 ± 0.00	29.53 ± 2.36	12.31 ± 7.40	24.98 ± 7.26
SPDL	20.87 ± 7.40	12.57 ± 2.41	27.47 ± 4.13	21.18 ± 6.56	8.69 ± 6.57
GC (Proposed)	3.96 ± 0.37	2.16 ± 0.45	0.64 ± 0.02	2.35 ± 1.11	7.30 ± 2.79

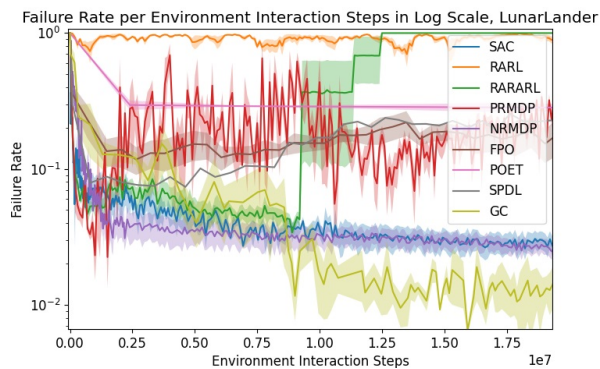


Fig. 5. Characteristic Training Curve from LunarLander Benchmark with Respect to Environment Interaction Steps

TABLE IV
REWARD, FAILURE RATE, AND MEAN GENETIC DISTANCE BETWEEN TRAINING EXAMPLES DURING ABLATION STUDY

Method	Reward	Failure Rate(%)	Genetic Distance
Base RL (SAC)	291.76	10.2	22.65
GC (Ours)	304.33	3.96	10.60
No Mutate	294.17	8.51	10.44
No Crossover	271.72	17.63	20.92
Random Failure	251.37	24.50	23.34
Single Run	99.45	33.33	0

the rate of performance improvement longer and converges to a better solution.

C. Ablation Study

One of the insights from Table IV is, except for Single Run, curricula with similar scenarios, i.e. shorter mean genetic distance, perform better. While Random Failure builds a curriculum with failed scenarios, the similarity between the scenarios is not ensured. In the case of No Crossover, genetic similarity between scenarios is low as unlike crossover which mixes sequences from two parents to generate two offsprings, mutation only creates one offspring from one parent. The

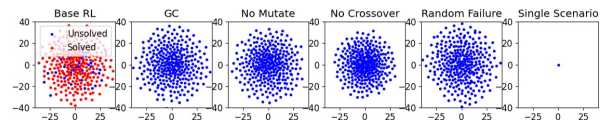


Fig. 6. tSNE analysis on genetic distance of generated scenarios

difficulty in transferring skills between more distant scenarios seems to result in No Crossover and Random Failure performs poorly.

Figure 6 on the other hand highlights how a coverage over scenario space affects performance for curriculums with short mean genetic distance between scenarios. As an agent is trained based on scenarios it experiences, having curriculum scenarios more spread out in the scenario space can help the agent generalize across diverse scenarios. While Single Run keeps genetic distance between scenarios to a minimum, it offers a poor coverage of the scenario space as in Figure 6. Genetic Distance between scenarios generated by No Mutate is similar to those in GC, but the former offers narrower coverage of the scenario space. While No Mutate can only reorganize genetic sequences it had upon initialization, GC can introduce new sequences through mutation, allowing it to explore a wider scenario space and train a more robust agent.

VIII. CONCLUSION AND FUTURE WORKS

This paper proposes genetic curriculum, an RL algorithm that uses a genetic algorithm to generate a curriculum of scenario for training RL agents. Through empirical study, our algorithms show improvement over existing state-of-the-art approaches concerning robustness. Future works will focus on decreasing the computational load of our algorithms, improving rate of convergence, as well as implementing our method in real and more complex benchmarks.

REFERENCES

- [1] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, “Solving rubik’s cube with a robot hand,” *arXiv preprint arXiv:1910.07113*, 2019.
- [2] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox, “Closing the sim-to-real loop: Adapting simulation randomization with real world experience,” in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8973–8979, IEEE, 2019.
- [3] R. Kaushik, T. Anne, and J.-B. Mouret, “Fast online adaptation in robotics through meta-learning embeddings of simulated priors,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5269–5276, IEEE, 2020.
- [4] N. Hansen, R. Jangir, Y. Sun, G. Alenyà, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang, “Self-supervised policy adaptation during deployment,” in *International Conference on Learning Representations*, 2020.
- [5] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.
- [6] L. Research, “X-plane 11.” <https://www.x-plane.com>, 2017.
- [7] G. N. Iyengar, “Robust dynamic programming,” *Mathematics of Operations Research*, vol. 30, no. 2, pp. 257–280, 2005.
- [8] A. Nilim and L. El Ghaoui, “Robust control of markov decision processes with uncertain transition matrices,” *Operations Research*, vol. 53, no. 5, pp. 780–798, 2005.
- [9] S. Mannor, O. Mebel, and H. Xu, “Lightning does not strike twice: robust mdps with coupled uncertainty,” in *Proceedings of the 29th International Conference on Machine Learning*, pp. 451–458, 2012.
- [10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in neural information processing systems*, pp. 2672–2680, 2014.
- [11] A. Kurakin, I. Goodfellow, S. Bengio, Y. Dong, F. Liao, M. Liang, T. Pang, J. Zhu, X. Hu, C. Xie, *et al.*, “Adversarial attacks and defences competition,” in *The NIPS’17 Competition: Building Intelligent Systems*, pp. 195–231, Springer, 2018.
- [12] P. Samangouei, M. Kabkab, and R. Chellappa, “Defense-gan: Protecting classifiers against adversarial attacks using generative models,” in *International Conference on Learning Representations*, 2018.
- [13] C. Xiao, B. Li, J.-Y. Zhu, W. He, M. Liu, and D. Song, “Generating adversarial examples with adversarial networks,” in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pp. 3905–3911, 2018.
- [14] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, “Robust adversarial reinforcement learning,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2817–2826, JMLR.org, 2017.
- [15] X. Pan, D. Seita, Y. Gao, and J. Canny, “Risk averse robust adversarial reinforcement learning,” in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 8522–8528, IEEE, 2019.
- [16] C. Tessler, Y. Efroni, and S. Mannor, “Action robust reinforcement learning and applications in continuous control,” in *International Conference on Machine Learning*, pp. 6215–6224, PMLR, 2019.
- [17] S. Paul, M. A. Osborne, and S. Whiteson, “Fingerprint policy optimisation for robust reinforcement learning,” in *International Conference on Machine Learning*, pp. 5082–5091, PMLR, 2019.
- [18] K. A. Ciosek and S. Whiteson, “Offer: Off-environment reinforcement learning,” in *Thirty-first AAAI conference on artificial intelligence*, 2017.
- [19] S. Paul, K. Chatzilygeroudis, K. Ciosek, J.-B. Mouret, M. A. Osborne, and S. Whiteson, “Alternating optimisation and quadrature for robust control,” in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [20] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, “Curriculum learning,” in *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.
- [21] W. Zaremba and I. Sutskever, “Learning to execute,” 2015.
- [22] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer, “Scheduled sampling for sequence prediction with recurrent neural networks,” in *Advances in Neural Information Processing Systems*, pp. 1171–1179, 2015.
- [23] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu, “Automated curriculum learning for neural networks,” in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1311–1320, JMLR.org, 2017.
- [24] M. P. Kumar, B. Packer, and D. Koller, “Self-paced learning for latent variable models,” in *Advances in Neural Information Processing Systems*, pp. 1189–1197, 2010.
- [25] L. Jiang, D. Meng, Q. Zhao, S. Shan, and A. G. Hauptmann, “Self-paced curriculum learning,” in *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.
- [26] M. Asada, S. Noda, S. Tawaratsumida, and K. Hosoda, “Purposeful behavior acquisition for a real robot by vision-based reinforcement learning,” *Machine learning*, vol. 23, no. 2-3, pp. 279–303, 1996.
- [27] A. Karpathy and M. Van De Panne, “Curriculum learning for motor skills,” in *Canadian Conference on Artificial Intelligence*, pp. 325–330, Springer, 2012.
- [28] D. Held, X. Geng, C. Florensa, and P. Abbeel, “Automatic goal generation for reinforcement learning agents,” 2018.
- [29] B. Ivanovic, J. Harrison, A. Sharma, M. Chen, and M. Pavone, “Barc: Backward reachability curriculum for robotic reinforcement learning,” in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 15–21, IEEE, 2019.
- [30] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, “Reverse curriculum generation for reinforcement learning,” in *Conference on robot learning*, pp. 482–495, PMLR, 2017.
- [31] C. Florensa, D. Held, X. Geng, and P. Abbeel, “Automatic goal generation for reinforcement learning agents,” in *International conference on machine learning*, pp. 1515–1528, PMLR, 2018.
- [32] S. Narvekar and P. Stone, “Learning curriculum policies for reinforcement learning,” in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 25–33, International Foundation for Autonomous Agents and Multiagent Systems, 2019.
- [33] P. Fournier, M. Chetouani, P.-Y. Oudeyer, and O. Sigaud, “Accuracy-based curriculum learning in deep reinforcement learning,”
- [34] P. Klink, C. D’Eramo, J. Peters, and J. Pajarinen, “Self-paced deep reinforcement learning,” in *NeurIPS*, 2020.
- [35] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, “Poet: open-ended coevolution of environments and their optimized solutions,” in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 142–151, 2019.
- [36] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, “Openai gym,” *arXiv preprint arXiv:1606.01540*, 2016.
- [37] B. Ellenberger, “Pybullet gymperium.” <https://github.com/benelot/pybullet-gym>, 2018–2019.
- [38] createmind, “Deep reinforcement learning.” <https://github.com/createamind/DRL/tree/master/spinup/envs/BipedalWalkerHardcore>, 2019.
- [39] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann, “Stable baselines3.” <https://github.com/DLR-RM/stable-baselines3>, 2019.
- [40] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.