

Robust Reinforcement Learning via Curricular Learning

Yeeho Song
December 13, 2024
CMU-RI-TR-24-75



The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania

Thesis Committee:

Advisor, Jeff Schneider (Chair)
David Held
Zachary Manchester
Jeff Clune, (University of British Columbia)

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Robotics.*

Abstract

Reinforcement Learning (RL) presents great promises for autonomous agents. However, when using robots in a safety critical domain, a system has to be robust enough to be deployed in real life. For example, the robot should be able to perform across different scenarios it will encounter. The robot should avoid entering undesirable and irreversible states, such as crashing into obstacles and ideally should meet the safety consideration even if its primary goals cannot be achieved.

One way to improve an RL agent’s robustness is to explore a variety of scenarios, environment parameters and opponent policies, via domain randomization. However, as an agent’s performance gets better, it becomes less likely to explore the region the agent performs poorly. An approach to solve this problem is Adversarial training, where an adversarial agent tries to inject noise to make the ego agent perform poorly. However, in such a setup, it’s much easier for the adversary to win over the ego agent, and hence ego agent often fails to overcome the adversarial noise without expert supervision. Also, as we move robots into more unstructured environments, environmental factors can affect the distribution of state space and dynamics more than what can be encoded as noise.

In my thesis, I will discuss how we can use curriculum learning to help an agent explore against variety of different situations, opponents, and dynamics efficiently to help achieve a robust performance. The first part of the thesis will introduce ideas in curriculum learning and how it can be used to explore a wide range of environment. The second part will be expanding such concept to multiagent domain and see how curriculum learning can help find a robust policy against collaborative and competitive, symmetrical and asymmetrical setups. Finally, I will be expanding the findings to the Quality Diversity domain and explore how the curriculum learning can help us to find a library of behaviors that cumulatively achieves robustness.

Acknowledgements

I would like to express my sincere gratitude for their part in my journey to this thesis.

First and foremost goes to my advisor, Professor Jeff Schneider. Professor Schneider has been very supportive throughout the program. He was always open to new ideas and approaches and eager to share insights from both well-established and novel problems. He has been a great mentor and a role model as a researcher and his guidance has been a great inspiration for my program.

I would also like to thank my comm members, Professors David Held, Zachary Manchester, and Jeff Clune. I thank them for their expertise and valuable feedback that improved my work and its presentation. The committee covers a diverse set of backgrounds essential for developing robust autonomous agents, and their insights were truly remarkable.

I also thank the members of the Auton Lab and the CMU community, Jack Good, Mononito Goswami, Chase Noren, Dominic Guri, Brian Yang, Ian Char, Viraj Mehta, Swapnil Pande, Conor Igoe, Ben Freed, Youngseog Chung, Tejus Gupta, Ceci Morales, Nick Gisolfi and Angela Chen. I thank them for the insightful discussion as well as mental support over the years. Thanks to them, the Auton Lab and CMU have truly felt like home.

Finally, I would like to thank my family back home. I am not the first person in my family to do a PhD program and I think this has somehow inspired me that there is so much to explore and discover outside of what's already written in textbooks. I thank them for their life-long inspiration and support.

Contents

1	Introduction	1
1.1	Advancements in Artificial Intelligence and Need for Robustness	1
1.2	Approach for Addressing Robustness	2
1.3	Our Approach for Robustness	3
2	Robustness With Respect to Environment	5
2.1	Introduction	5
2.2	Related Works	6
2.3	Background	8
2.3.1	Problem Statement	8
2.4	Approach	9
2.5	Experiments	11
2.5.1	Benchmarks	11
2.5.2	Baseline Algorithms	12
2.5.3	Evaluation and Hyperparameters	12
2.5.4	Ablation Study	14
2.6	Results	14
2.6.1	Comparison with Baseline Algorithms	14
2.6.2	Comparison with Respect to Environment Interactions	16
2.6.3	Ablation Study	17
2.7	Conclusion	18
3	Robustness With Respect to Opponent Policies	19
3.1	Introduction	19
3.2	Related Works	20
3.3	Approach	22
3.3.1	Preliminaries	22
3.3.2	Problem Formulation	22
3.3.3	Genetic Algorithm for Curriculum Generation	23
3.3.4	GenOpt Agent and Scenario Space	24
3.4	Experiments	25
3.4.1	Benchmarks	25

3.4.2	Baseline Algorithms	26
3.4.3	Evaluation and Hyperparameters	27
3.5	Results	28
3.5.1	Round Robin Results	28
3.5.2	Evolution of Generated Curriculum	29
3.5.3	Ablation Study	29
3.6	Conclusion	31
3.7	Limitations and Future Works	31
3.8	Additional Results : Against OpenAI’s Agent	31
4	Robustness By Quality-Diversity	33
4.1	Introduction	33
4.2	Related Works	34
4.3	Approach	35
4.3.1	Preliminaries	35
4.3.2	Our Approach	36
4.4	Experiments	38
4.4.1	Benchmarks	38
4.4.2	Baselines	41
4.4.3	Evaluation and Hyperparameters	42
4.5	Results	44
4.6	Conclusion	45
5	Conclusion	46
6	Appendix A. Additional results and implementation details for Chapter 3	47
6.1	Full Performance Results of the Trained Algorithms	47
6.2	Environment Details	47
6.2.1	Pong	50
6.2.2	Volley	50
6.2.3	ACM	51
6.2.4	Ant	52
6.2.5	SimpleAdversary	52
6.2.6	Environment Encoding	53
6.3	Implementation Details	53
6.3.1	Training Hardware	53
6.3.2	Tuning RL Algorithm	53
6.3.3	RL Hyperparameters	55
6.4	Baseline Hyperparameter Tuning	56
6.5	Crossover and Mutation	57
6.6	Sampling Parents	57
6.7	Evolution of Curriculum	58

List of Tables

2.1	Training Duration and Testing Resolution	13
2.2	Hyperparameter Search on Baseline Algorithms	13
2.3	Reward and Mean Failure Rate of Trained Agents(%)	15
2.4	Reward, Failure Rate, and Mean Genetic Distance between Training Examples during Ablation Study	17
3.1	Mean Win:Tie:Lose Ratio (%) of algorithms against baseline algorithms and ours (<i>GEMS</i>). In bold are the statistically significant highest win rates and lowest lose rates.	28
3.2	Mean return of the ablation study in ACM benchmark.	30
4.1	Hyperparameter Sweep for the SAC algorithm used in this chapter. Selected value in bold.	42
4.2	Hyperparameter Sweep for the SAC algorithm used in this chapter. Selected value in bold.	43
4.3	Cumulative Performance of the Trained Algorithms	43
6.1	Mean return of algorithms against baseline algorithms and ours for Pong, Volley, and ACM. The highest mean return in bold.	48
6.2	Mean return of algorithms against baseline algorithms and ours for Ant and SimpleAdversary. The highest mean return in bold.	48
6.3	Detailed Win:Tie:Lose Ratio(%) of Each Algorithm Against Others.	49
6.4	Summary of environmental parameters.	54
6.5	Selected hyperparameters for RL algorithms.	55
6.6	Selected hyperparameters for baselines.	56

List of Algorithms

1 Genetic Curriculum (GC)10 2 Genetic Multi-agent Self-play24 3 Curricular Quality
Diversity38

List of Figures

1.1	Figure of a table summarizing safety challenges with respect to application	2
2.1	Curriculum generation during POET(115). Unlike GC, POET runs training inside genetic algorithm, greatly increasing computational load.	7
2.2	Visualization on crossover and mutation. GC 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.	10
2.3	Screenshot of Benchmarks used in this chapter (from left to right), BipedalWalker(Hardcore/System), LunarLander, Hopper, and Walker which tests agent’s robustness against a variety of obstacle courses / actuator failure.	11
2.4	Training Curve for BipedalWalkerHardcore. Lower the better	15
2.5	Characteristic Training Curve from LunarLander Benchmark with Respect to Environment Interaction Steps	16
2.6	tSNE analysis on genetic distance of generated scenarios	17
3.1	Overview of our proposed approach: During training, the ego student agent is trained against a scenario describing the choice of opponent, action script for the GenOpt Agent, and the choice of environment parameters. Our curriculum records regret and win/lose/tie outcomes for each scenario. Based on the performance of the scenarios in the previous population, our curriculum generator uses a genetic algorithm to generate a new population of scenarios to be used to train the agent for an epoch at each fixed interval of steps.	20
3.2	Visualization of crossover and mutation. Crossover replaces encoded segments between two parent scenarios. Mutation changes an encoded segment of a parent with a random sequence.	23
3.3	Screenshots of Pong, Volley, ACM (dots marking trajectories), Ant, and SimpleAdversary	25

3.4	Evolution of scenarios generated by our GEMS. Each dot marks the position of red and blue aircraft at 1-second intervals. The color of the markers starts from black to blue for student aircraft and black to red for the opponent aircraft.	29
3.5	Ablation study training curve	30
3.6	Training curve of the GEMS in Ant benchmark evaluated against OpenAI’s agent. Left shows performance in terms of cumulative reward. Right shows performance in terms of win, lose and tie.	32
4.1	Visualization of the 4X rules. Upon initialization, each behavior (visualized as red and blue in this figure) is assigned a random curricular cell. During exploration and expansion, the curriculum distribution is allowed to spread into bordering cells. During extermination cells, the cells contested for expansion by more than one behavior, and cells bordered by cells of different behaviors (Marked with question marks) are evaluated and assigned to the best-performing behavior.	37
4.2	Visualization benchmarks used in this chapter. (From left to right) BipedalWalker, LunarLander, and Ant	39
4.3	Training curve of the baselines across benchmarks. (From left to right) BipedalWalker, LunarLander, and Ant	44
6.1	Training curve with round robin results.	47
6.2	Each dot marks the position of the red and blue aircraft at 1-second intervals. The color of the markers transitions from black to blue for the student aircraft and from black to red for the opponent aircraft. At the start of the training, only our GEMS can present scenarios and opponents that provide interesting data points instead of simply crashing to the ground. While SPDL+FSP somewhat succeeds in finding scenarios that do not end in a crash, it optimizes scenarios without considering opponent policy. It simply finds trivial cases where agents are flying in circles.	59

Chapter 1

Introduction

1.1 Advancements in Artificial Intelligence and Need for Robustness

Artificial Intelligence (AI) and Deep Learning have experienced significant growth and have shown great potential in recent years. Several companies are introducing AI agents capable of recognizing and making suggestions based on data (79; 80), which are already starting to improve our quality of life. There are also efforts to introduce agents capable of not only making suggestions to humans when they see certain data but also knowing how to physically interact with the world and execute it when they see certain data. Regarding agents capable of making decisions on their own, Reinforcement Learning (RL) shows great promise. Rather than requiring an expert to handcraft and hardcode what actions to take per each situation, an RL agent can autonomously train itself by exploring and interacting with real or simulated environments. RL agents have already shown better than human performances in various simulated environments, such as Go, racing, and even air-to-air combat (92; 104; 118)

Such application also raises a new challenge of robustness as summarized by Figure 1.1. While agents can demonstrate a super-human performance, such agents will not be deployed should the human operators not trust the safety of the agents. For example, gaming AI is relatively well-trusted and deployed in real life. Even if the gaming AI encounters a situation where it cannot deliver its full performance and fails to win the game, the results are not catastrophic. The goal of a game is to have fun, and as long as the human users enjoy the experience, a gaming AI that occasionally loses the game is acceptable. Likewise coding assistants such as the Open AI's Copilot is relatively well-trusted and deployed. While the agent will make coding suggestions, it is ultimately the human programmer who will accept the recommendations and run the program. Assuming that the human programmer does his or her job, the agent's failure will not directly lead to a catastrophic failure. On the other hand, in the case of autonomous driving, autonomous agents are the decision-making algorithms



Program	Gaming AI	Coding AI	Autonomous Driving
	 <small>Mastering the Game of Go with Deep neural Networks and Tree Search, Silver et al, 2016</small>		 <small>Argo AI</small>
Function	Play a Game	Recommend Code	Drive a Car
Can Result in Irreversible and Undesired State?	✗	✗	○

Figure 1.1: Figure of a table summarizing safety challenges with respect to application

that are often the last piece of an autonomy pipeline that interacts with the real world. Autonomous driving agents are designed with the goal of removing safety-driver altogether and therefore, should an autonomous agent make a failure, it will lead to a failure without humans to filter the errors and the results can be catastrophic and sometimes irreversible depending on the situation. Therefore, to deploy autonomous agents in real life for safety-critical applications, we not only need the agent to perform well but also do so consistently across various situations and environments without resulting in irreversible and catastrophic states. This thesis will approach robustness as a way to ensure that the agent will not end up in an irreversible and catastrophic state in the situations it is expected to operate in.

1.2 Approach for Addressing Robustness

Research for robustness in machine learning comes in many ways. Some focus on verification that a trained model will not fail in the operating dataset (40; 55). Extending this concept to the RL, one can use Control Barrier Functions (CBF) (3) to ensure that the RL policy will not venture into irrecoverable states (30). CBFs are designed with expert knowledge of the domain and dynamics to define which states are safe and recoverable.

In conjunction with this idea, we focus on how wide situations can an autonomous agent cover and solve without entering irreversible and catastrophic zones. In this thesis, we focus on robustness as an agent’s capability to achieve high performance in a wider field of situations. We aim to train robust policies capable of being deployed into safety-critical situations such as driving and flying where the inability to handle some of the unavoidable long-tail scenarios prevents the deployment of RL agents.

1.3 Our Approach for Robustness

One of the main issues for robustness in RL is that training against the last 10% of the situations is often significantly more difficult than training against the first 90% of the situations. As an agent’s performance gets better and better it is less likely to encounter a situation in which it does not do well, resulting in inefficient exploration of the problem domain. In addition, as training progresses, an agent moves closer and closer to its local minima, and adding a new skill becomes more difficult as an agent approaches convergence. To achieve robust performance, an agent should have a way to explore and gather data better than just randomly exploring the environment.

In conjunction with the above problem, our thesis will be focusing on the use of curriculum learning (10) for training robust policies for safety-critical applications. Chapter 2 will introduce the main ideas in curricular learning and genetic algorithms and show how such ideas can be incorporated to train an RL agent to be robust against a diverse set of environments and dynamics. The included empirical study shows how such an approach can result in agents x2.8 less likely to fail a task compared to state-of-the-art baselines.

Aside from differences in environment, the huge variety of opponents agents can face during deployments presents a huge challenge for training an agent capable of handling a variety of situations. For example, when driving a car, an agent must solve against a variety of different road conditions, car dynamics, and other vehicles’ behaviors. A robust agent should be capable of learning a general policy that applies to a wide variety of situations and dynamics, as well as coping with the training instability when trying to learn against such a breadth of training examples.

Chapter 3 will expand the idea into the multiagent domain and show how curricular RL can be used to train an RL agent to be robust against a diverse set of opponent policies. The included empirical study shows how such an approach can outperform the state-of-the-art approaches in both competitive and collaborative games, symmetrical and asymmetrical. To the best of our knowledge, we are the first approach to solve the Open AI AntSumo (6) benchmark by training from scratch without pre-trained weights or expert supervision.

Lastly, one of the unavoidable issues with any policy is that it is difficult for a policy to be completely free of failure modes in which the policy is not well-trained. Sometimes, this could arise from a trade-off between performance and safety where meeting both criteria is conflicting. In some cases, it could be that the situation is so rare that was not properly considered nor included during the training phase. In real life, instead of proving robustness against the unpredicted, robustness is often achieved by redundancy and multiple backup systems. For example, in the case of the aerospace industry, should aircraft exit normal operating boundaries predefined by the primary flight controls, an aircraft will switch to alternative control law. This delegates more control authority to the pilots who provide redundancy and backup in situations not fully covered by the avionics computer.

Inspired by such an approach, Chapter 4 will expand the idea of curriculum RL to quality-diversity, where we train a population of multiple policies with different behaviors to cover a wider area of the scenario space. The included empirical study shows that such an approach can cover a wider range of benchmarks and scenarios while minimizing expert supervision required in previous approaches.

Chapter 2

Robustness With Respect to Environment

2.1 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 section, We develop 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 this 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.

2.2 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; 16; 46; 60) and high fidelity simulators (26; 96) 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 (54; 70; 78) and deep learning architectures (41; 64; 98; 119), 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) (90) uses an adversary to apply external force to the protagonist. Risk averse robust adversarial RL (RARARL) (82), probabilistic action robust Markov decision process (MDP) and robust action robust MDP (PRMDP / NRMDP) (112) 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. GC 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) (87) shares insights on encoding environments and scenarios. Building upon the previous works on classical RL (18) (86), 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. GC 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, (10), curricular learning has shown success in supervised learning tasks (9; 43; 56; 63; 121) as well as various RL tasks (4; 35; 36; 37; 51; 53; 59; 76). Automatically generating a curriculum of similar yet gradually more complex scenarios is an ongoing question

in curriculum learning. Self-paced deep RL (SPDL) (62) 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 explore non-parametric approaches to expand the ideas to non-fixed raw encoding of curriculum.

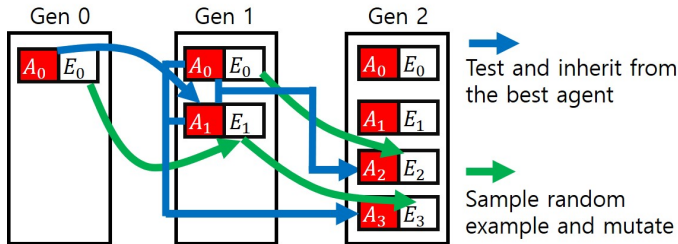


Figure 2.1: Curriculum generation during POET(115). Unlike GC, POET runs training inside genetic algorithm, greatly increasing computational load.

Paired open-ended trailblazer (POET) (115) borrows several elements from genetic algorithms to use non-fixed length encodings. As shown in Figure 2.1, 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. GC 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. GC 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.

2.3 Background

This chapter 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 $p_{train}(\psi)$;

$$\pi^* = \underset{\pi}{\operatorname{argmax}} \sum_{\psi} p_{train}(\psi) J_\psi(\pi) \quad (2.1)$$

2.3.1 Problem Statement

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

$$G_\psi(\pi) = \begin{cases} 0, & \text{if } \pi \text{ fails for } \psi \\ 1, & \text{otherwise} \end{cases} \quad (2.2)$$

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;

$$\pi_{robust}^* = \underset{\pi}{\operatorname{argmax}} \sum_{\psi} p_{test}(\psi) G_\psi(\pi) \quad (2.3)$$

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. I, therefore, take the curricular approach of investigating how $p_{train}(\psi)$ can be better selected to train a robust agent.

In other words, the utility of a policy J comes from the reward function of the environment and this is the optimization goal of the RL algorithm. On the other hand

G is for our safety objective and is coming from the boolean definition of whether a policy has solved the scenario or not, and this is the optimization goal or the fitness function of our curriculum generator. With the curriculum generator oversampling the difficult cases using G and the RL algorithm trying to optimize its performance using J against the given curriculum, we try to optimize our policy based on our robustness goal of solving more and more scenarios.

2.4 Approach

To train a robust agent, We select training scenarios as scenarios the agent currently fails in. Solving these examples directly addresses Equation (2.3). 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 the scenarios will help an agent learn not only a specific task but also a variety of similar tasks as well. We explore 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 none-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 . This provides our fitness function.

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 2.2. 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 2.2.

Once $|\Psi_{population}| \geq M_{pop}$, GC exits the scenario generation cycle and $\Psi_{training}$ is used to train π for an epoch. Policy training can be done by any RL algorithm. For example, the BipedalWalker benchmarks (13) used Soft Actor Critic (19) as Train Agent. As for the reward function given to the RL algorithm, we used the default reward functions of the environments we implemented. Algorithm 2 shows the pseudocode of GC.

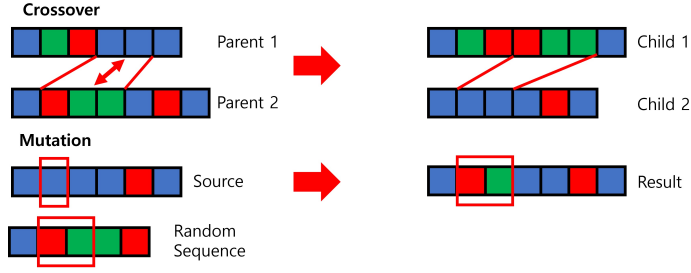


Figure 2.2: Visualization on crossover and mutation. GC use genetic algorithm to use raw, non-fixed length encoding to generate similar scenarios that can act as a curriculum and dynamically change length of encoding to keep up with agent’s performance.

$$p_{parent}(\psi) = \begin{cases} \frac{1}{(\max(L(i)) - L(i) + 1)}, & \text{if } \pi \text{ fails } \psi \\ 0, & \text{otherwise} \end{cases} \quad (2.4)$$

Algorithm 1 Genetic Curriculum (GC)

```

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

```

GC 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 2.2, 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 (87). Another feature visible in Figure 2.2 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.

2.5 Experiments

2.5.1 Benchmarks

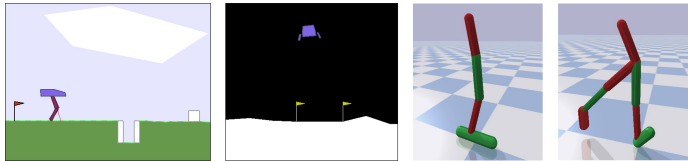


Figure 2.3: Screenshot of Benchmarks used in this chapter (from left to right), BipedalWalker(Hardcore/System), LunarLander, Hopper, and Walker which tests agent’s robustness against a variety of obstacle courses / actuator failure.

BipedalWalkerHardcore (13) 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 (13). 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 (29). 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, I 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.

2.5.2 Baseline Algorithms

Some of the state of the art approaches have been chosen as follows. To compare GC against adversarial RL approaches, I chose RARL (90), RARARL (82), and PRMDP / NRMDP (112). I chose FPO (87) for comparison against approaches that control the training environment. For comparison against curricular RL, I chose SPDL (62) for parametric curricular approaches and POET (115) for non-parametric approaches.

The algorithms in this chapter 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) (19), while Hopper and Walker use twin delayed DDPG (TD3) (94). 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 (61) and activated with the ReLU function.

2.5.3 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 GC 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 this chapter is concerned about how robust a converged solution is, I 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, this chapter 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.

This chapter 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, I report the performance of the best agent in terms of reward as the result of the random seed. Table 2.1 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 2.2. 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, I 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.

Table 2.1: 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 2.2: 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

The default versions of the benchmarks random use number generators to create scenarios at each run. For FPO and SPDL, I engineered a fixed-length encoder where the range of the numbers coming out of the random number generator was defined.

For POET and GC 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.

2.5.4 Ablation Study

To better understand how GC 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.

2.6 Results

2.6.1 Comparison with Baseline Algorithms

As shown in Figure 6.1 and Table 3.1, 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 3.1 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

Table 2.3: 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 (Ours)	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 (Ours)	3.96 ± 0.37	2.16 ± 0.45	0.64 ± 0.02	2.35 ± 1.11	7.30 ± 2.79

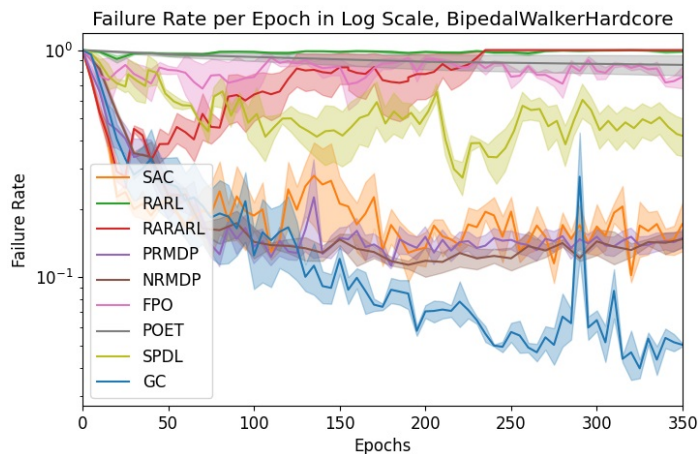


Figure 2.4: Training Curve for BipedalWalkerHardcore. Lower the better

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 the 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.

2.6.2 Comparison with Respect to Environment Interactions

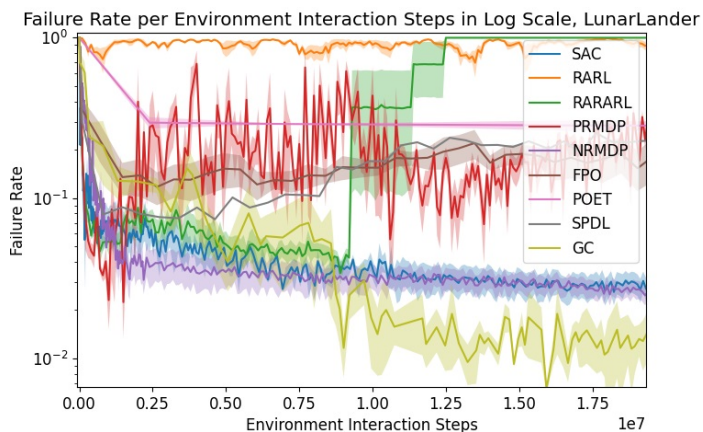


Figure 2.5: Characteristic Training Curve from LunarLander Benchmark with Respect to Environment Interaction Steps

One of the important criteria in RL is how efficiently it can learn per the number

of environment interactions. Figure 2.5 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 the rate of performance improvement longer and converges to a better solution.

2.6.3 Ablation Study

Table 2.4: 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

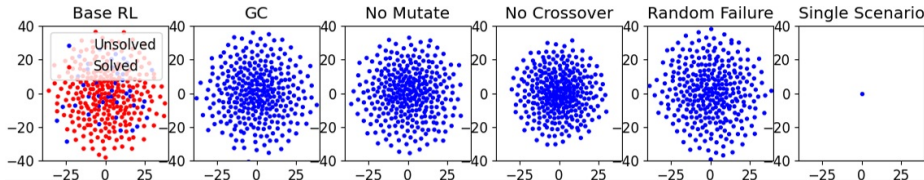


Figure 2.6: tSNE analysis on genetic distance of generated scenarios

One of the insights from ablation studies in Chapter 2.1.7 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 difficulty in transferring skills between more distant scenarios seems to result in No Crossover and Random Failure performs poorly.

Figure 2.6 on the other hand highlights how a coverage over scenario space affects performance for curricula 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 2.6. 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.

2.7 Conclusion

This chapter explores genetic curriculum, an RL algorithm that uses a genetic algorithm to generate a curriculum of scenario for training RL agents. Through empirical study, the algorithms show improvement over existing state-of-the-art approaches concerning robustness. For supplementary videos to this chapter, please visit <https://sites.google.com/andrew.cmu.edu/robustreinforcementlearning>

Chapter 3

Robustness With Respect to Opponent Policies

3.1 Introduction

Competitive multi-agent reinforcement learning (RL) has gained interest for its potential in various fields, such as gaming, robotics, finance, and cybersecurity, where agents need to outperform others. To discover new competitive strategies, self-play is often used to explore the environment with RL agents. Most self-play algorithms train ego agents against a population of opponent policies to avoid overfitting to a specific opponent. (49; 50; 114).

When training against a population of policies, selecting the policies most useful for training is challenging. Some works use game-theoretic methods to select opponents (65; 77), but finding approximate Nash Equilibria in setups with large search spaces is costly. Others use curriculum learning to select opponent policies (99), but relying on random sampling to find candidates is not scalable with large search spaces. Therefore, we propose using genetic algorithms, which have proven efficient for searching and generating scenarios with large scenario spaces in single-agent domains (108).

Another issue with self-play is training instability due to the adversarial vulnerability of RL policies (67; 97; 120). RL agents often overfit to a specific opponent’s weakness, resulting in non-generalizable behaviors. For example, when training against a mujoco-ant agent trained to play sumo (7), agents would learn to win by waving their legs to confuse the opponent, instead of the non-trivial solution of pushing the opponent. Exploitation destabilizes self-play, especially during the initial phases when the opponent’s policy is underdeveloped. Initiating opponents with expert demonstrations (114) to avoid this issue can be costly. We, therefore, introduce GenOpt Agents, which are environment-agnostic open-loop agents optimized by genetic algorithms to match the performance of the ego agent.

To summarize, our approach utilizes genetic algorithms to efficiently search and generate an optimal training curriculum in expansive search spaces, while GenOpt

Agents improve training stability. An ablation study provides insights into our methodology’s effectiveness and design choices. Figure 3.2 summarizes our approach. Supplementary information and codes can be found in <https://github.com/yeehos/GEnetic-Multiagent-Selfplay.git>

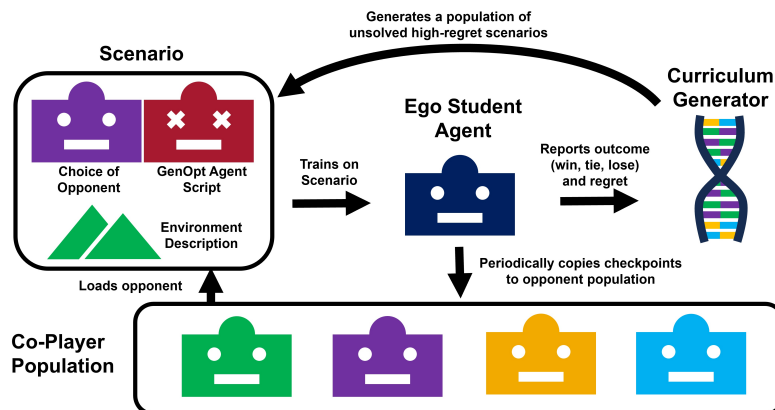


Figure 3.1: Overview of our proposed approach: During training, the ego student agent is trained against a scenario describing the choice of opponent, action script for the GenOpt Agent, and the choice of environment parameters. Our curriculum records regret and win/lose/tie outcomes for each scenario. Based on the performance of the scenarios in the previous population, our curriculum generator uses a genetic algorithm to generate a new population of scenarios to be used to train the agent for an epoch at each fixed interval of steps.

3.2 Related Works

Game Theoretic Approach to Self-Play

Training an RL agent in competitive multi-agent environments requires an opponent. Unlike rule-based opponents (15) or expert demonstrations (17), self-play (103; 111) agents to autonomously explore and discover competitive behaviors. However, training against only one RL opponent can suffer from the instability of opponent behavior changing over time and ego agent overfitting to a specific policy (38). Therefore, Fictitious Self-Play (FSP) (14; 49; 66) proposes to train against a population of opponents. In deep RL, this opponent library is often compiled by saving checkpoints of the ego policy during training (12; 114).

Expanding on these findings, game theoretic approaches like Policy Space Response Oracle (PSRO) (12; 65; 114) utilize Nash equilibrium (77), a stable point in a multi-agent game where no player can unilaterally change their strategy to improve their payoff. PSRO calculates an approximate Nash strategy to determine the mixture of

checkpoints to load as opponents. However, computing an approximate Nash strategy can be computationally intensive if the policy space is large. Moreover, effectively learning policies in complex, non-stationary, and partially observable multi-agent environments remains a significant challenge. While some approaches use ensemble learning to split the learning task (106) (105), they are limited to simpler, discrete action setups. This chapter approaches this issue with a curricular viewpoint.

Curricular Reinforcement Learning

When it comes to learning a policy for a difficult problem, Curricular RL suggests that agents can learn faster by first being exposed to simple tasks and scenarios and gradually progressing to similar but slightly more difficult tasks. (2; 22; 25; 35; 36; 53; 62; 69; 76; 93).

Various papers explore curriculum optimization in the single-agent domain, such as Bayesian Optimization (88), teacher agents (27), and genetic algorithms (52; 115; 116). Genetic operations make scenarios similar to each other, thus achieving curriculum learning by facilitating the transfer of skills. Many genetic approaches rely on mutations, which make small alterations to the encoding of a scenario, in order to populate a curriculum with similar scenarios.

On the contrary, the Genetic Curriculum (GC) (108) uses population-wide genetic operations, such as crossover, for curricular RL. Crossover involves merging sequences encoding across a population, helping the transfer of skills within the curriculum by enhancing similarity among the scenarios. While successful in generating an effective curriculum in a large scenario space, GC requires costly evaluation steps to validate scenarios. Additionally, GC cannot regulate scenario difficulty.

One way to regulate difficulty is to use regret, which quantifies the gap between the current obtained return with the maximum possible return on the scenario. Showing success in various single-agent domains (57; 84), regret has been applied to curriculum learning in a multi-agent setup by Multi-Agent Environment Design Strategist for Open-Ended Learning (MAESTRO) (99). While MAESTRO introduced optimization of both environmental parameters and opponent selection to guarantee robustness, it is limited by relying on domain randomization to uncover new scenarios. Studies in GC (108) suggest this can lead to suboptimal performance. Unlike those generated by population-wide genetic operations, scenarios generated by domain randomization may not be similar to each other, making transfer of skills difficult and learning slow. We, therefore, combine both population-wide genetic operations with regret regulation for multi-agent curriculum generation.

Open-Loop Opponents for Multi-Agent Self-Play

Self-play can be unstable, especially at the beginning of training when the opponent is not well-trained to make effective moves in the game. While some approaches involve hand-crafted opponents (114) or agents trained via imitation learning from expert data (117), such supervision can be costly.

No-OP (No Operation) agents (109), which do not take any actions, can reduce the need for expert supervision. However, their simple and limited behavior limits the

approach to certain types of environments. For instance, a walking robot will fall if there is no torque in the joints, while a plane with no control input will eventually crash by losing speed and altitude due to drag. Our GenOpt addresses such issues by introducing agents that take optimized actions.

3.3 Approach

3.3.1 Preliminaries

An RL problem setup is typically represented as a tuple in a Markov decision process: $[S, A, P, r, \gamma]$, where S is the state space of a problem, A is the action space, P is the transition dynamics, r is the return of a state-action pair, and $\gamma \in [0, 1)$ is the temporal discount factor. The agent’s policy, $\pi(a | s)$, maps states $s \in S$ to actions $a \in A$. The utility of a policy π is the expected return is denoted as $J(\pi) = \mathbb{E} \sum t \gamma^t r(s_t, \pi(s_t))$. During training, an RL algorithm optimizes the policy with respect to the data it has collected.

In our multi-agent setup, we consider that the utility of a policy is dependent on both the opponent’s policy π_{opp} and the environment ψ , denoted as $J(\pi_{ego}, \pi_{opp}, \psi)$. At each epoch consisting of a fixed number of time steps, we save the current version of our ego agent and add it to the library of possible opponents to choose from Π_{opp} . We describe the environment with parameters, ψ . We define a scenario ξ as an opponent policy - environment pair: $\xi = \{\pi_{opp}, \psi\}$. The goal of our curriculum generator at a given timestep is to generate a population of scenarios, $\Xi_{train} = \{\xi_0, \xi_1, \dots, \xi_n\}$, where optimizing the ego policy π_{ego} with respect to the curriculum Ξ_{train} will result in a policy with the behaviors we desire.

$$\pi^* = \max_{\pi} J(\pi_{ego}, \Xi_{train}) \tag{3.1}$$

3.3.2 Problem Formulation

For the multi-agent setup, we want our algorithm to continuously explore the game by visiting scenarios that it does not do well to learn new competitive behaviors. We design our curriculum generator around this idea by first formulating multi-agent learning as a zero-sum game $G(\pi_{ego}, \xi)$ between the π_{ego} and the curriculum generator. $G(\pi_{ego}, \xi) = 1$ if π_{ego} wins the scenario ξ , $G(\pi_{ego}, \xi) = 0$ if it was a tie, and $G(\pi_{ego}, \xi) = -1$ if loses. The opponent π_{opp} and the environment parameters ψ are determined by the curriculum’s scenario ξ . The solution for a finite zero-sum two player game is minimax;

$$G^* = \min_{\Xi} \max_{\pi_{ego}} G(\pi, \Xi) \tag{3.2}$$

Therefore, our curriculum generator will generate a population Ξ that minimizes $G(\pi_{ego}, \Xi)$

3.3.3 Genetic Algorithm for Curriculum Generation

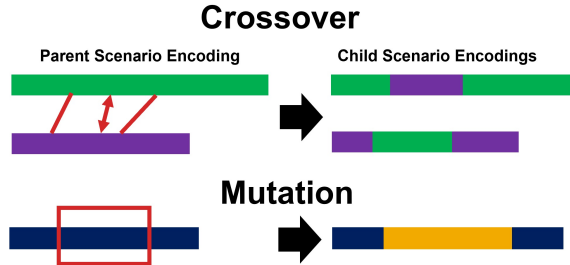


Figure 3.2: Visualization of crossover and mutation. Crossover replaces encoded segments between two parent scenarios. Mutation changes an encoded segment of a parent with a random sequence.

In this section and onward, we explain how our proposed algorithm, GENetic Multi-agent Self-play (GEMS), as described in Algorithm 2, generates curriculum.

To generate a curriculum, GEMS uses a genetic algorithm due to its proven efficiency in large scenario spaces and its flexibility in scenario encoding. At the beginning of the training, as in Line 5 of Algorithm 2, GEMS randomly initializes curriculum Ξ_{train} . π_{ego} trains on the Ξ_{train} for an epoch consisting of a fixed number of steps. For each scenario, our algorithm records whether the agent has won, lost, or tied, along with the regret estimated by positive value loss (57; 58).

At the end of each epoch, GEMS harvests scenarios from the current curriculum to generate the next curriculum as in line 14 of Algorithm 2. The fitness function, $p(\xi)$, defines the probability of a scenario being harvested. Since the GEMS is trying to minimize $G(\pi_{ego}, \Xi)$ of the generated curriculum, fitness is proportional to $(1 - G(\pi_{ego}, \xi))$. At the same time, GEMS needs to generate scenarios with high information potential, measured as regret, $\delta(\xi)$. Therefore, fitness is set as $p(\xi) \propto \delta(\xi)(1 - G(\pi_{ego}, \xi))$.

From these harvested scenarios, GEMS uses crossover and mutation (see ??) to create an offspring population consisting of sequences of ξ similar to the ones harvested. Crossover occurs when the algorithm takes a random segment from one parent scenario encoding and swaps it with a random segment from another parent scenario encoding. The mutation is when the algorithm selects a random segment from a parent scenario encoding and swaps it with a segment from a randomly generated scenario encoding. Scenarios ξ s generated by the genetic algorithm will inherently be similar, aiding in

the transfer of skills and, consequently, a faster rate of convergence when used as a curriculum. Detailed operations for crossover and mutation are shown in Section 6.5.

Unlike GC, which has separate evaluation steps, GEMS calculates fitness from the performance measured during training, reducing the computational cost. In addition, GEMS regularizes difficulty.

3.3.4 GenOpt Agent and Scenario Space

Algorithm 2 GENetic Multi-agent Self-play

```

1: Initialize Policy  $\pi_{ego}$ 
2: #Select size of curriculum,  $M_{train}$ 
3: #Select mutation rate,  $p_\mu$ 
4: Input  $M_{train}, p_\mu$ 
5: Initialize Curriculum  $\Xi_{train}$ 
   while True do
   |   o
   |   end
   |   outcome = [] regrets = []
6: #Train  $\pi$  with  $\Xi_{train}$  by exploring scenario  $\xi$ 
   for  $\xi$  in  $\Xi_{train}$  do
7:   end
   |    $G(\pi, \xi), \delta = \text{Train}(\pi_{ego}, \xi)$ 
8:   outcome.append( $G(\pi_{ego}, \xi)$ )
9:   regrets.append( $\delta$ )
10:
11: #Harvest Examples
12:  $\Xi_{seed}, \text{utility} = \text{harvest}(\Xi_{train}, \text{outcome}, \text{regret})$ 
13: #Generate New Curriculum
14:  $\Xi_{train} = \text{crossover}(\Xi_{seed}, M_{train}, \text{utility})$ 
15:  $\Xi_{train} = \text{mutation}(\Xi_{train}, p_\mu)$ 
16: save( $\pi_{ego}$ ) #Regularly save checkpoint
17:

```

Self-play in deep RL often suffers from training instability due to ego agents exploiting the adversarial vulnerability of the opponents. While using open-loop agents can mitigate this issue, previous approaches used expert supervision to design open-loop behaviors, which can be costly (109). To allow an open-loop agent that continuously evolves to match the ego agent’s performance during training without expert supervision, we introduce GenOpt Agent, denoted as π_\emptyset , that optimizes open-loop behaviors using genetic algorithms without supervision.

$\pi_\emptyset = \{(t_0, a_0), (t_1, a_1), (t_2, a_2), \dots\}$ encodes the opponent as a list of non-fixed lengths describing which actions to take in an open-loop fashion. For example, at

timestep $t_1 < t < t_2$, the opponent takes action a_1 . As genetic algorithms can be agnostic to the length of encoding, the encoding length does not need to be pre-specified and is optimized by the curriculum generator without expert supervision. GenOpt’s open-loop nature makes it less vulnerable to opponent-induced confusion, but unlike No-OP, GenOpt is competitively optimized to match the ego agent’s performance, enhancing its applicability across diverse environments.

Using the features above, GEMS encodes a scenario as $\xi = \{i_\pi, \pi_\emptyset, \psi\}$ i_π is an integer representing the i -th checkpoint to load as the opponent π_{opp} . If $i_\pi = 0$, it indicates that no agent will be loaded, and instead, a GenOpt Agent π_\emptyset will be used as the opponent. ψ denotes which environment parameters to load. It should be noted that whether it is used or not, the encoding for the GenOpt Agent π_\emptyset is always present in a scenario ξ . This ensures that the optimization and the memory about a GenOpt Agent π_\emptyset are not lost during the genetic operations and can always be brought back if needed throughout the training.

3.4 Experiments

3.4.1 Benchmarks

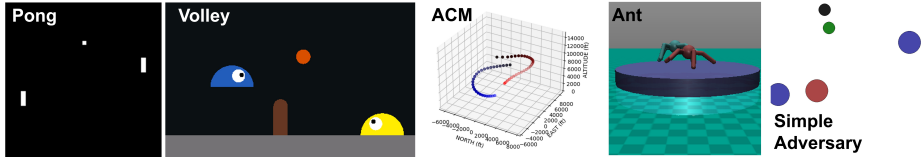


Figure 3.3: Screenshots of Pong, Volley, ACM (dots marking trajectories), Ant, and SimpleAdversary

We evaluate our algorithm in Pong, Volley, ACM, Ant, and SimpleAdversary to assess its performance in different levels of complexity. More environment details are in Section 6.2.

Pong is a 2-player, continuous-action space version similar to Atari Pong (13). Each player controls a paddle to hit a ball back and forth, aiming to score by getting the ball past the opponent. The scenario set the players’ sides and the ball’s initial velocity. Simple 1-D movement and fixed horizontal ball speed focus make dynamics easy to learn and relatively emphasize the game-theoretic aspect of the solution more than other benchmarks.

Volley is based on (47). Each player uses continuous action input to control an avatar, moving left, right, and jumping. The goal is to bounce the ball to land on the other side of the map across the net. The scenario specifies the players’ sides and the

initial ball velocity. Volley incorporates gravity, elastic collisions, and a net, increasing complexity and skill requirements.

ACM is a simulated dogfighting environment. Each player flies an airplane in 3D space, aiming to position its nose toward the opponent without crashing to the ground. The scenario describes the spawning airplanes’ positions, postures, and velocities. With physics simulated by JSBSim (11), a high-fidelity simulator widely adopted in autonomous aircraft and aircraft controls research due to its accurate aerodynamic modeling (91), and direct control over the aircraft operating in 3D space with no stability assist, ACM is a complex environment that needs well-trained skills to master the game with non-linear dynamics. Additionally, since the game’s objective is to maneuver and position the ego plane in relation to the opponent, the scenario, encoding spawn locations and orientations of the planes, significantly influences the game’s outcome. ACM tests whether algorithms are robust when environmental factors outside their control give an unfair advantage to one of the agents.

Ant is a simulated Mujoco environment based on OpenAI’s Competitive Gym environment. (7). The game’s goal is to flip the opponent or push the opponent out of the ring. Ant challenges the agent to learn multiple different skills to master the game. For example, while ramming is often the only way to win an opponent that stands still, those who ram are vulnerable to those who flip the opponent. However, flipping is not helpful against opponents that stand still, as an opponent keeping all its legs on the ground leaves no room for flipping. The agents must learn various skills and know how to mix and use them to win a game. To the best of our knowledge, we are the first paper to solve the Ant problem by training from scratch without expert supervision or pretrained weights.

SimpleAdversary is a Multi Particle Environment from (110). As in the Figure 2.3, the environment features one adversary (red), two good agents (blue), and two landmarks (green and black). Agents are rewarded based on their proximity to the target landmark, with opposing goals for good agents and the adversary. Only good agents know the target’s location. SimpleAdversary tests offers asymmetrical environment with goals, action and observation space being different per role, along with added challenge of collaborative-competitive settings with multiple agents.

3.4.2 Baseline Algorithms

We use PSRO as one of the baselines for comparison. While many algorithms have emerged from PSRO, such as Rectified PSRO (5), Pipeline PSRO (71), or Anytime PSRO (73), these are fundamentally based on PSRO’s framework. They share the underlying structure of generating Nash strategy opponents without considering the ego agent’s performance to present the tasks in a gradual, easy-to-learn fashion. Therefore, we utilize PSRO as a characteristic example to highlight the limitation of this assumption. While PSRO necessitates additional steps to evaluate each policy to run the meta-solver during training, we report performance without considering the extra

evaluation steps that PSRO requires. We believe that presenting PSRO’s performance without factoring in these additional steps provides a comprehensive comparison, not only against PSRO itself but also against newer subsequent algorithms such as XDO (72) and NAC (33), which aim to reduce the computational cost associated with policy evaluation.

We include GC (108) as a comparison against approaches that use a genetic algorithm to generate a curriculum. We include SPDL (62) to compare our approach against algorithms controlling environment parameters and actively regulating difficulty levels during curriculum generation. For fairness in the multi-agent domain, GC and SPDL are running with Fictitious Self-Play (FSP) (49), labeled GC+FSP and SPDL+FSP, where the curriculum generator can choose opponents from saved checkpoints. While other single-agent curricula RL regulate difficulty levels, such as using regret, we decide to use an approach more relevant to the multi-agent domain by including MAESTRO (99). MAESTRO represents a state-of-the-art approach to optimizing environment-opponent choices in the multi-agent domain using regret to regulate the difficulty level of the scenarios. Finally, we include FSP as a baseline comparison for simple population-based multi-agent RL. FSP loads environmental parameters by domain randomization.

For all algorithms, we use a publicly available implementation of Soft Actor Critic (SAC) (19; 45) and Proximal Policy Optimization (PPO) (8; 101) as the base strategy explorers.

3.4.3 Evaluation and Hyperparameters

We conduct experiments to evaluate each method’s effectiveness using ten random seeds for Pong, Volley, and 5 for the more complex ACM and Ant benchmarks. The training duration varied from 5 to 15 days, with computationally intensive algorithms like ACM requiring the most time. Given the impracticality of exploitability analysis in complex environments, performance was assessed through competitions against five baselines and our GEMS across 200 games with randomized environment parameters, utilizing Round Robin format for both interim (every $2.5e5$ timesteps for SimpleAdversary, every $5e5$ timesteps for the rest) and final evaluations. Ablation studies were performed with five seeds to validate our design decisions further, comparing ablated models against fully trained baselines.

While PSRO, GC, and SPDL involve additional simulation steps for curriculum generation, we report results based on the exploration steps taken by each agent for easier comparison. We train each algorithm for $7e6$ steps in each benchmark, and $3.5e6$ steps for the SimpleAdversary. See Section 6.2 for implementation details and hyperparameter tuning results.

	Pong	Volley	ACM	Ant	SimpleAdversary
FSP	59 : 1 : 39	40 : 5 : 55	37 : 49 : 14	38 : 28 : 34	50 : 0 : 50
PSRO	56 : 1 : 42	40 : 5 : 55	35 : 51 : 14	37 : 21 : 42	53 : 0 : 47
GC+FSP	58 : 1 : 42	46 : 7 : 47	23 : 34 : 43	36 : 24 : 39	49 : 0 : 51
SPDL+FSP	31 : 0 : 69	47 : 24 : 28	28 : 43 : 29	37 : 21 : 42	46 : 0 : 54
MAESTRO	21 : 0 : 79	42 : 6 : 52	10 : 23 : 67	34 : 22 : 43	49 : 0 : 51
GEMS	72 : 3 : 25	48 : 24 : 28	41 : 52 : 8	44 : 36 : 21	53 : 0 : 47

Table 3.1: Mean Win:Tie:Lose Ratio (%) of algorithms against baseline algorithms and ours (*GEMS*). In bold are the statistically significant highest win rates and lowest lose rates.

3.5 Results

3.5.1 Round Robin Results

As summarized by the round robin results in Table 3.1, Our approach outperforms all baselines in the win and lose rates across benchmarks. See Section 6.1 for comprehensive results.

While MAESTRO performs comparable to FSP in Volley, it does not perform well in other benchmarks. GC (108) suggested that relying on random exploration instead of curriculum generation can be limiting. We will confirm that this limitation extends to the multi-agent setup in our ablation study.

While PSRO performs well in Pong, where dynamics are simple and approaching game-theoretical solutions is relatively more important, PSRO’s effectiveness diminishes in more complex environments like Volley, ACM, and Ant. This highlights the significance of a curriculum that aids in learning new skills against a broad spectrum of opponents. Also, while PSRO’s game theoretic approach allows it to outperform most of the baselines in an asymmetrical setup such as SimpleAdvesary, as shown in Table 6.2, our GEMS outmatches PSRO via flexible architecture applicable to such setups as well.

Despite both being curricular methods, GC+FSP and SPDL+FSP exhibit different performances. SPDL+FSP can regularize scenario difficulty and outperforms GC+FSP in more complex environments such as Volley, ACM, and Ant compared to PONG. GC+FSP, on the other hand, has a robustness guarantee unlike SPDL+FSP, and outperforms SPDL+FSP in simple benchmarks such as Pong, where difficulty regularization is not important. However, both suffer from ACM benchmark as neither are designed for multi-agent setups.

MAESTRO’s performance is comparable to FSP in Volley but is less effective in other benchmarks, especially in ACM and Ant. This highlights the limitations of relying on random exploration in expansive scenario spaces. Our ablation study further supports this finding in the multi-agent context.

3.5.2 Evolution of Generated Curriculum

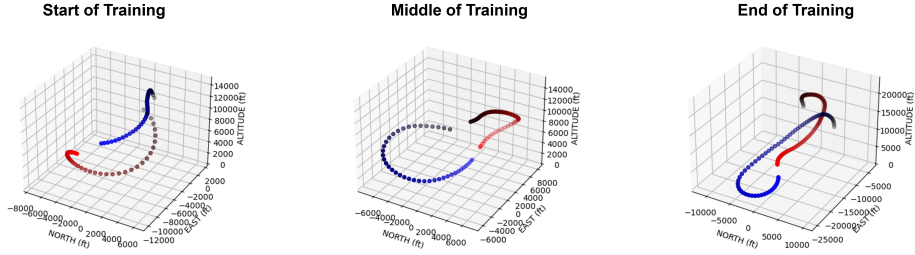


Figure 3.4: Evolution of scenarios generated by our GEMS. Each dot marks the position of red and blue aircraft at 1-second intervals. The color of the markers starts from black to blue for student aircraft and black to red for the opponent aircraft.

To demonstrate the curriculum evolution, Figure 3.4 shows characteristic scenarios generated at 1e6, 3e6, and 7e6 steps with the ACM benchmark. Initially, to accommodate the untrained ego agent, our algorithm generates GenOpt that spawns close and circles around the ego agents, offering non-trivial and easy opponents for the ego agent to practice basic tracking. As the training progresses, our GEMS starts to generate scenarios with interesting learning examples, such as the opponent chasing the ego agent for a head-on pass. This gradual complexity increase enables the agent to tackle complex situations by training’s end, such as a much longer chase shown in the figure. For extensive visualizations and comparisons, see Figure 6.2 in the appendix.

3.5.3 Ablation Study

We conduct an ablation study on our GEMS to demonstrate the empirical effects of our design choices. **NoRegret** experiment excludes the regret term (δ) when calculating the fitness function to observe how effectively regret regulates the difficulty level of scenarios during training. **NoGenetic** experiment employs the approach of MAESTRO (99) instead of using a genetic algorithm to generate a scenario population. Scenarios are individually selected from a replay buffer, and new scenarios are added through a random search, deviating from the batch generation of a genetic algorithm. **NoCrossover** experiment disables the crossover function of the genetic algorithm, relying solely on mutation for scenario search. **NoGenOpt** experiment disables the option to use a GenOpt Agent during training. **NoVic** experiment disables the fitness function from utilizing $G(\pi_{ego}, \psi)$, focusing solely on maximizing regret. Lastly, **No-OP** replaces GenOpt Agents with No-OP agents, which takes no actions.

While NoRegret initially mirrors GEMS in training performance, it plateaus earlier. This marks that while regret becomes crucial for adjusting curriculum difficulty towards

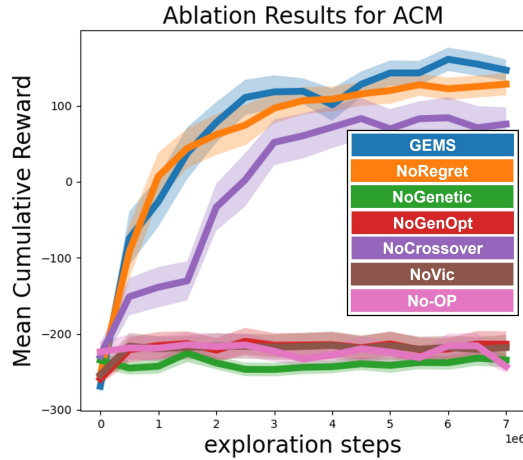


Figure 3.5: Ablation study training curve

MAESTRO	-226.845 ± 25.332
GEMS	139.080 ± 32.768
NoRegret	128.416 ± 14.724
NoGenetic	-234.64 ± 9.595
NoCrossover	75.808 ± 21.963
NoVic	-218.267 ± 16.605
NoGenOpt	-213.751 ± 16.907
No-OP	-243.172 ± 8.521

Table 3.2: Mean return of the ablation study in ACM benchmark.

equilibrium, its impact on training efficacy is less significant compared to GEMS’s other design features.

One key aspect of GEMS is the use of genetic operations. GC (108) showed that genetic algorithms outperform random exploration in single-agent settings by efficiently identifying challenging scenarios similar to each other, facilitating effective task generation. The ablation study shows that this advantage also applies to the multi-agent domain, where even solely relying on mutation (NoCrossover) is more effective than random exploration (NoGenetic).

However, beyond difficulty regularization by regret, GEMS has several useful features for multi-agent training. One such is the GenOpt Agent, which provides non-trivial open-loop agents without expert supervision. Compared to the No-OP approach, which involves agents taking no actions, the ablation study shows that training with GenOpt is better. In the ACM environment, for instance, an uncontrolled aircraft’s inevitable energy loss leads to a crash. The GenOpt Agent, on the other

hand, addresses this issue by providing a sophisticated sequence of actions optimized by the curriculum generator, thereby ensuring sustained engagement and challenge.

Finally, NoVic underscores the critical role of Nash equilibrium in our zero-sum game formulation between the curriculum generator and the ego agent. Unlike NoRegret, NoVic exhibits inferior performance, illustrating that while regret contributes to curriculum generation, $G(\pi_{ego}, \psi)$ is pivotal for ensuring training stability and effective curriculum optimization.

3.6 Conclusion

This chapter proposes using a genetic algorithm to improve and stabilize learning in multi-agent environments. Using the strength of genetic algorithms to generate a curriculum in a large scenario space, we enable an RL agent to reach a better solution faster. We also introduce GenOpt Agent, an open-loop agent optimized by the curriculum generator without expert supervision. This enhances training stability, which is especially valuable early on when the self-play opponent lacks sufficient competence. Empirical results across multiple benchmarks show our method outperforming various baselines, and an ablation study further confirms the significance of our design decisions. For supplementary videos to this chapter, please visit <https://sites.google.com/andrew.cmu.edu/robustreinforcementlearning>

3.7 Limitations and Future Works

Similar to MAESTRO (99), our method currently lacks a provable guarantee for convergence to the Nash equilibrium in games. This presents an opportunity to identify conditions under which guaranteed convergence can be achieved.

3.8 Additional Results : Against OpenAI’s Agent

When the Ant benchmark was first released, OpenAI trained agents with manual curriculum and self-play to play as opponent for those who will use the benchmark for their research. When comparing our agent to OpenAI’s agent, we report 50.9% victory for GEMS, 37.2% for OpenAI, and 11.9% tie. We have uploaded the codes as well as the trained weights for our best performing model as a new benchmark for people to use. Figure 3.6 visualizes the training curve of our training. Note that GEMS never saw OpenAI’s agent during training and evaluation was done with fixed saved weights after the training was completed.

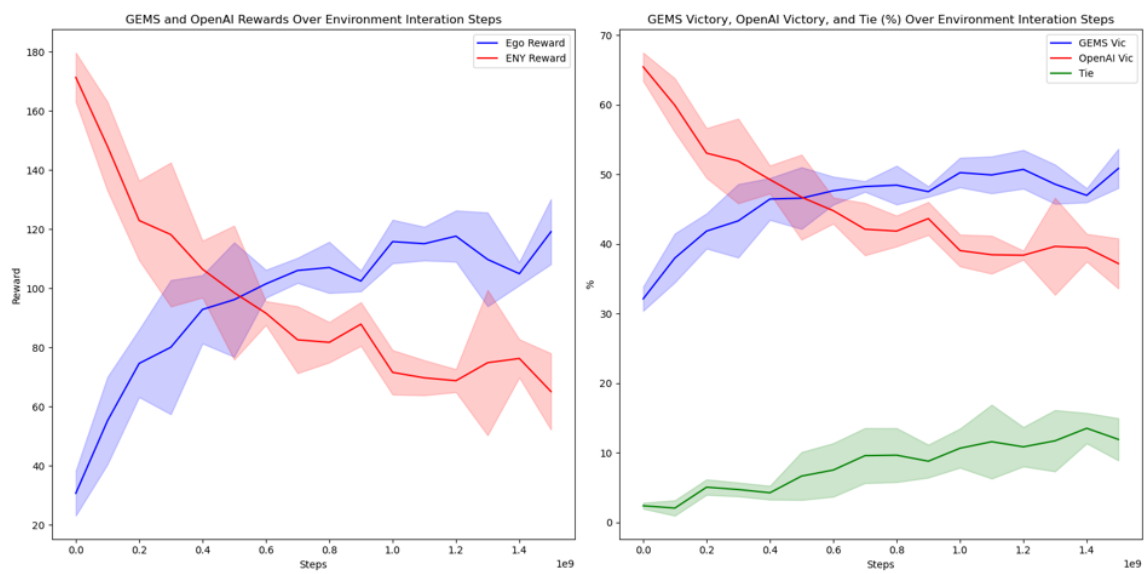


Figure 3.6: Training curve of the GEMS in Ant benchmark evaluated against OpenAI’s agent. Left shows performance in terms of cumulative reward. Right shows performance in terms of win, lose and tie.

Chapter 4

Robustness By Quality-Diversity

4.1 Introduction

In recent years, there has been an increased interest in applying reinforcement learning (RL) for complex systems requiring a high level of robustness due to strict operating conditions intolerable for failure, from nuclear fusion reactors (24; 74; 102) running billions of dollars worth of scientific equipment to autonomous driving (32) operating cars in proximity with people. With such environments having little margin for error, having an agent that covers a variety of situations and finds a reasonable control solution is crucial. However, as with all controllers, RL-based controllers have some crucial design considerations that have to be addressed in order for the controller to be applied to real-world problems.

First, it is difficult to have a controller that can fit all situations with optimal performance. In machine learning, there are some problems where it is empirically shown that for some problems there is no single universally best method that solves every problem and outperforms all other baselines (42; 83; 100). Likewise, in the controls and RL domain, it is difficult to have a single controller to achieve optimal performance in every situation. Sometimes, this can be due to conflicting optimization criteria (performance vs safety), or difficulty in covering the entire situation with a single controller.

Second, all controllers will inevitably have a weak point where they perform less optimally compared to other situations they encounter. RL is known to have adversarial weakness (39; 44; 81; 107), and given enough time and resources, such weakness may be found. This could be an adversary agent actively trained to find the weakness or simple environment perturbation that pushes the agent out of its designed operating area.

As such weakness is present in any system, robustness in real life is often achieved by redundancy and backups. For example, modern airliners use on-board flight computers to generate actual command input to the aircraft's control surfaces. While flight computers are designed to be robust, there are reported cases where this

procedure no longer works due to aircraft inadvertently entering a flight regime which the flight computer was not designed for, or the flight computer itself having some malfunctions. However, instead of such incidents being fatal, airplanes would generally have redundancy and backups. For example, when the primary flight computer is no longer working, the aircraft would switch to alternate flight control mode. In this mode, the aircraft would not be flying as optimal and fuel efficiently as if aided by the flight computer, but the pilot would have more control authority over how the control surfaces deflect to steer the aircraft and recover the aircraft. While it is possible, however unlikely, that the primary flight computer can fail, the probability of both primary flight computer, backup flight computer, and human pilots failing all at once is so low that flying is often regarded as one of the safest modes of transport.

Inspired by such a design, we explore how we can improve the robustness of RL controllers. By having multiple different agents with different performance, weaknesses, and strengths, we aim to deliver higher performance across a wider situation an autonomous agent can encounter during deployment. In this regard, we look into Quality-Diversity, where we train multiple policies with different but well-performing behaviors to populate our library of possible controllers to be selected during deployment. We incorporate ideas from curriculum learning and to efficiently explore the policy space to optimize the combined performance of the policy library. Unlike existing approaches in Quality-Diversity where expert supervision is required to design a metric for diversity, we instead take inspiration from nature where diversity arises not from manual design, but due to difference in situations the agents are optimizing for (23).

4.2 Related Works

When it comes to changing or switching between behaviors to better fit the given situation, Meta-RL (28) is one approach where agents will change behavior based on what they experience during test time. In such approaches, agents will update their controller to better fit the data it sees and collects during test time. While in theory, this will provide infinitely many kinds of behaviors optimized for every situation it encounters, such approaches assume unlimited resets and cheap cost of collecting data, which is not feasible in safety-critical applications, Single-Life Reinforcement Learning (17; 85) tries to mitigate this issue by not resetting the system during test time, but such approaches for exploration and data collection purposes still assumes that all states are recoverable and does not result in catastrophic failure, which is not feasible in safety-critical systems. Moreover, real-world systems often do not have enough computation resources to update entire neural networks while exploring at the same time.

In this regard, Probabilistic Embeddings for Actor-critic meta-RL (PEARL) (95) offers a more viable option. Using a conditional network structure, the policy with trained with ground truth information description of the situation as a latent condi-

tional variable. During the test time, this latent variable would be estimated using a separate classification or a regression module. Such structure allows for an infinite variety of different behaviors depending on how the latent variable is encoded and has a relatively small search space to be fitted during test time instead of updating the entire neural network. However, such an approach requires expert supervision to represent the situation in latent variables during training and is limited to benchmarks where a clear latent encoding can be used. For example, if the latent encoding has no useful spatial description of the environment, the conditional network will simply ignore the latent embedding and try to fit all situations at once.

On the other hand, some approaches go full-expert supervision to differentiate and diversify between behaviors. In the case of Agile But Safe (48), the algorithm uses two different policies with different behaviors to control a quadruped robot. When the situation is nominal, the nominal controller optimized for performance takes over. When the robot loses stability, a safety controller optimized for stability recovery takes over and moves the robot back into a nominal state. While this allows the robot to efficiently navigate through unstructured terrain using two different behaviors, such behaviors had to be carefully prepared by expert supervision with customized reward functions and goal states.

In contrast to the above case where diversity was achieved by expert supervision, Diversity is All You Need (DIAYN) (31) achieves diversity by self-supervision. Instead of relying on an engineered reward function, DIAYN gives rewards by self-supervision by using a classifier trained to differentiate the behaviors being trained online. While achieving great success in skills discovery, DIAYN’s behaviors are designed for diversity, not performance. Therefore, DIAYN behaviors are not directly applicable to solving problems and require a supervisory agent that stitches the discovered skills to make a coherent working policy.

In this regard, Quality-Diversity approaches both the quality and diversity of the trained set at the same time. MAP-Elites (20) and the expanded framework in the deep-RL domain, Actor-Critic Quality Diversity Reinforcement Learning (ACQDRL) (34; 68; 89) uses the idea of quality-diversity to train behaviors that are different from each other but also optimal in their respective occupancy in diversity space. However, the quality-diversity approached based on the MAP-Elites framework requires expert domain knowledge and supervision to design a metric that can differentiate one behavior from another.

4.3 Approach

4.3.1 Preliminaries

An RL problem setup is typically represented as a tuple in a Markov decision process: $[S, A, P, r, \gamma]$, where S is the state space of a problem, A is the action space, P is the transition dynamics, r is the return of a state-action pair, and $\gamma \in [0, 1)$ is the temporal

discount factor. The agent’s policy, $\pi(a | s)$, maps states $s \in S$ to actions $a \in A$. The utility of a policy π is the expected return is denoted as $J(\pi) = \mathbb{E} \sum t\gamma^t r(s_t, \pi(s_t))$. During training, an RL algorithm optimizes the policy concerning the data it has collected. In a single behavior RL, we only have to consider a single policy π_0 representing a single behavior.

$$\pi^* = \max_{\pi} J(\pi, \Xi_{train}) \tag{4.1}$$

In multi-behavior RL, we consider the utility of the population of behaviors Π against the scenario space. While selecting which behavior to use during test time would be dependent on some algorithm that takes care of test-time adaption, we will assume that we can choose the policy representing the optimal behavior π_{ξ} at a given test-time scenario ξ , for simplicity. Using this oracle model, we can define the utility of a behavior population as the sum of performances across scenario space paired with best-performing behaviors as in Equation (4.2).

$$\pi_{\xi} = \max_{\pi} J(\pi, \xi), \pi_{\xi} \in \Pi \tag{4.2}$$

With this formulation, our goal would be to find the best-performing population of behaviors, Π^*

$$\Pi^* = \max_{\Pi} = \sum_{\xi \in \Xi} J(\pi_{\xi}, \xi) \tag{4.3}$$

During training, the behavior will be optimized for the scenarios it is being trained for as follows;

$$\pi_{\xi} = \max_{\pi} J(\pi, \xi) \tag{4.4}$$

Using Equation (4.4) and Equation (4.2), solution to the Equation (4.3) can be found by assigning following curriculum;

$$\xi_{train, \pi_i} = \xi, \pi_i = \max_{\pi} J(\pi, \xi) \tag{4.5}$$

4.3.2 Our Approach

However, when using curricular learning, using Equation (4.5) is not feasible. For example, if two different behaviors are trained on two different distributions of scenarios with the mean of the distributions being close to each other, the resulting behaviors will also be similar to each other. Therefore, to achieve diversity, the distribution of the scenarios should be grouped so that the mean of the distributions are far away from each other.

Due to this requirement, we approach the curriculum generation with the 4X (eXplore, eXpand, eXploit, and eXterminate) algorithm, an argmax solution to multi-tangent domain often used in electronic games. Under the formulation, we start the

training by having each behavior first initialized in their respective cell in the scenario space. At each interval, as part of the explore and expand rules, the behavior can spread its curriculum to include the nearby cells. As part of the exploitation process, each behavior will exploit its curriculum by being trained against the scenarios in the curriculum cells to improve their performance in their assigned cells. As part of the extermination process, should a contested for expansion by more than one behavior, or should a cell be bordered by a cell occupied by a different behavior, the curriculum generator will evaluate the performance of each behavior and assign the cells to the behavior that performs better in the cells in question.

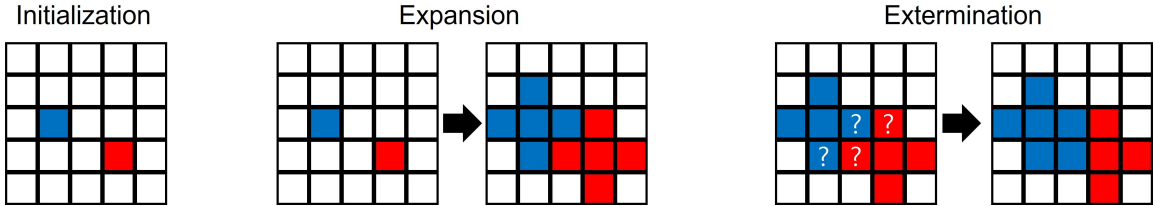


Figure 4.1: Visualization of the 4X rules. Upon initialization, each behavior (visualized as red and blue in this figure) is assigned a random curricular cell. During exploration and expansion, the curriculum distribution is allowed to spread into bordering cells. During extermination cells, the cells contested for expansion by more than one behavior, and cells bordered by cells of different behaviors (Marked with question marks) are evaluated and assigned to the best-performing behavior.

The benefit of the 4X approach is that it conserves the argmax formulation in Equation (4.5) while also allowing for each behavior to extend and contract their coverage via specialization - generalization trade-off. For example, if an agent covers more area than it can cover, it will start to lose the border cells to the nearby behavior that is more specialized in the zone. In this process of taking over the cells, the Nash Equilibrium is a point where none of the behaviors can take over nearby cells without losing performance and specializations in the cells it controls. As such, the equilibrium point of the behavior is where the global utility of the population is maximized. Assuming that the scenario space is not too complex, we can say that our approach will maximize the total welfare of the population.

One of the downsides of this approach is that each contested and bordering cell must be evaluated to decide which behavior to assign the cells to. As this can be quite costly, we replace the evaluation part with a regression model and only evaluate the scenario cells regression model is having difficulty assigning behavior to. For the case of experiments in this thesis, we used KNN models and limited actual evaluations to be less than 10% of the steps the agent interacted with the environment.

To summarize our algorithm, our algorithm works as follows. First, we randomly initialize b behaviors and segment the scenario space into kernels. We then randomly

assign starting cells to each behavior in which the curriculum generation can take place. We then sample random scenarios from the assignments Υ to generate a curriculum of a desired size. After each behaviors are trained on and exploited in their respective scenarios, we run the exploration and expansion steps to assign more curriculum cells to each behavior. When contested and bordering cells are generated, we take the extermination steps to reassign cells to each behavior. We continue the cycle again until we are out of iteration steps. Algorithm 3 shows the pseudocode for our proposed algorithm, Curricular Quality Diversity (CQD).

Algorithm 3 Curricular Quality Diversity

```

1: Initialize  $b$  Behaviors  $\pi_0, \dots, \pi_{b-1}$ 
2: Initialize  $b$  Assignments  $v_0, \dots, v_{b-1}$ 
3: #Select size of curriculum,  $M_{train}$ 
4: Input  $M_{train}$ 
   while True do
     end
5: #Exploit : Train Behaviors with Assignments
   for  $i$  in range( $b$ ) do
6:
   end
    $\xi_i = \text{GenerateCurriculum}(v_i)$ 
7:  $\pi_i = \text{Train}(\pi_i, \xi_i)$ 
8:
9: #Explore and Expand for  $i$  in range( $b$ ) do
10:
   end
    $\text{Assignment}_i = \text{ExploreAndExpand}(v_i)$ 
11:
12: #Exterminate
13:  $\Upsilon = \text{Exterminate}(\Upsilon)$ 
14:

```

4.4 Experiments

4.4.1 Benchmarks

As shown in Figure 4.2, we choose the following four benchmarks to demonstrate the characteristics of our proposed approach.

BipedalWalker

BipedalWalker is the same as the BipedalWalkerHardcore in Chapter 2. The agent observes the environment through IMU, joint encoders, and LIDAR on the body and

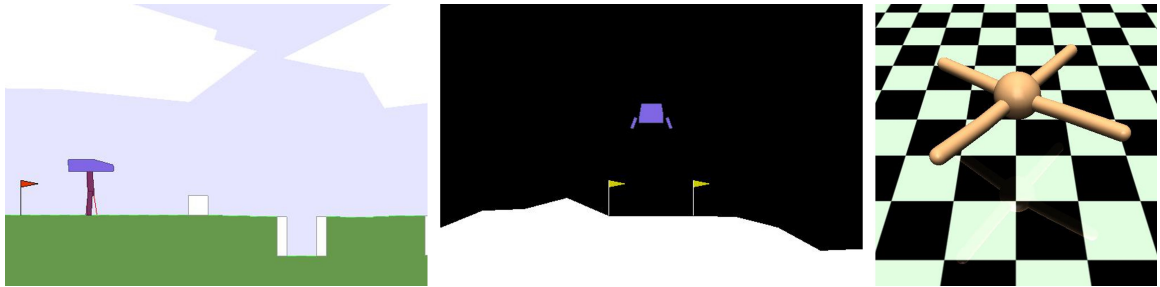


Figure 4.2: Visualization benchmarks used in this chapter. (From left to right) BipedalWalker, LunarLander, and Ant

controls its knee and hip joints to traverse through various obstacles. A scenario consists of various combinations of obstacles and grass fields in various ordering.

BipedalWalker provides a unique challenge as it represents the unforeseeable nature of failure cases. While there are states from which BipedalWalker agents are physically incapable of recovering, knowing where the unsafe states are does not solve the robustness problem as most well-trained policies will easily avoid those states in the first place. In the rare moments in which the agent enters such states, it is most likely not because the agent intentionally entered such states, but more likely due to the environmental factors, a sequence of critical events and obstacles, that the agent ended up in such states. Therefore, to avoid unsafe states, agents must prepare well before it starts to become unstable and lose energy by traversing through the obstacles beforehand. However, as LIDAR has a limited range the agent cannot see what the situation will be like. Moreover, the arrangement of the obstacles ahead the robot is not dependent on the ordering and obstacles behind the robot. Therefore, it is difficult to predict what challenges lie ahead until the robot is standing in front of the obstacles themselves. Such a feature of the BipedalWalker makes it a good representation of how the agent will be able to prepare against unforeseeable failure situations.

LunarLander

LunarLander is a benchmark similar to the LunarLander benchmark in the Chapter 2 but with even larger variation in failure modes, and higher fuel cost to compare and contrast the effect of two conflicting optimization goals between safety and performance (fuel usage). While the observation space is the same as in the Chapter 2, it has one additional degree of freedom for rotating the main engine gimbal. This is to give an alternative mode of controlling the lander’s orientation should failure with the RCS be too great to allow for a safe landing. The goal of the LunarLander is to safely land on the surface using its RCS thrusters, main engine gimbal, and main engine

thrust while minimizing fuel usage. Scenario space consists of various failure modes, limiting the actuation range of RCS, gimbal, and main engine throttle depending on the situation. The actuation range can be limited to 100% of the nominal to 30% of the nominal depending on the scenario.

LunarLander provides two unique challenges to the environment. First, due to the conflicting optimization goals between performance and safety, it is difficult to have a single behavior fit all situations. As mentioned in the previous chapters, the optimal policy for LunarLander is to free fall as much as possible and fire its main engines to a maximum at the last possible moment to minimize fuel usage. However, this method of performing a suicide burn can be risky if the main engine is unable to deliver as much thrust as it is needed. For such cases, the engines will have to start firing much earlier to compensate for the loss in thrust. However, this will lead to increased fuel usage and associated costs. Therefore, unless the dynamics and the failure mode of the scenario are given to the agent, there is no single behavior that can optimize performance while optimizing safety, and the scenarios must be distributed against multiple agents with different levels of focus on safety and performance.

Second, LunarLander pushes the curriculum generator whether it is capable of dividing scenario space the way it should be to diversify the behaviors. In the case of LunarLander, the scenario space consists of failure to the main engine, main engine gimbal, right RCS, and left RCS. However, the only variable that leads to variations in behavior is the magnitude of the main engine failure. For example, when trained with two different scenarios, the policies will exert the same behavior if the variation in main engine failure is the same. However, even if the variations in the main engine gimbal or RCS failure are different, the policies will show different behaviors if the magnitude of the main engine failure is different. Therefore, a good curriculum generator should be able to find how a scenario can be split to give more diversity to each policy and their respective behaviors.

Ant

Ant is a benchmark simulated by the MuJoCo physics engine (113). The agent has control of each of the ant-shape robot’s eight joints. The robot is free to move in the 3D space on a flat surface and the agent is rewarded if the ant robot moves towards the right. Borrowing ideas from legged locomotion research, diversity between different modalities of behavior and gaits is relatively well-understood and because of this, much quality-diversity research includes some form of multi-legged locomotion as part of the benchmark. The scenario-space for the Ant benchmark consists of failure modes, limiting how much power each motor can deliver in a given scenario. The actuation range can be limited to 100% of the nominal to 50% of the nominal depending on the scenario.

4.4.2 Baselines

We compare our approach with the following baselines inspired by various approaches emphasizing diversity in the behavior of the population.

Conditional

In our Conditional baseline, the policy is represented as a conditional network with latent conditional encoding describing the situation. Both during training and testing, this latent conditional encoding comes from expert supervision with access to ground truth information. As such, our Conditional baseline represents the maximum performance of the PEARL (95) based approaches.

In the case of LunarLander, and Ant benchmarks, this latent embedding comes from the ground truth information of what the failure situation is, which is how much power is limited to each of the joints and actuators. In the case of BipedalWalker, ground truth information is a combined discrete and continuous embedding of undefined length, and feeding a conditional network with raw data would be difficult. Instead, we condense the information to represent the percentage of each obstacle on the map. As the agent requires a long look ahead to avoid catastrophic failure, we believe that this is a good balance between performance and cost to tell an agent how aggressively or safely an agent should walk to maximize performance while maintaining safety.

Manual

In our Manual baseline, we have 4 behaviors in a behavior library, and each behavior is trained with a manually designed curriculum. In the case of LunarLander, behaviors are trained on different scenarios ranging from no failure to the main engines, maximum main engine power limit of 75%, maximum main engine power limit of 50%, and maximum main engine power limit of 30%. In the case of Ant, the behaviors are trained for a case where all legs are working, all legs are limited to 50% power and two different combinations of failure modes. In the case of BipedalWalker, we selected four random obstacle courses, with one of the courses being free of obstacles. Our Manual represents approaches such as ABS (48), where expert domain knowledge is used to train policies with different behaviors geared for different goals and situations.

Ensemble

In our Ensemble baseline, we have 4 behaviors in a behavior library and each behavior is trained on a randomly generated curriculum. This provides a baseline comparison of not having any curricular considerations in a multi-behavior setup.

	Swept Values
learning rate	1e-3, 3e-4 , 1e-4
γ	0.95, 0.98, 0.98
α	auto

Table 4.1: Hyperparameter Sweep for the SAC algorithm used in this chapter. Selected value in bold.

DIAYN

DIAYN baseline is our implementation of DIAYN (31) for the benchmarks. We keep the number of behaviors to four as with other baseline implementations.

MAP-ELITES

MAP-ELITES is our implementation of the MAP-ELITES baselines (21) for the benchmarks. Based on the original publication and following publications on Quality Diversity (34; 68; 89), we use the duty cycle, which is the ratio of time each leg spends in contact with the ground as the diversity metric for the BipedalWalker, and Ant. In the case of LunarLander, we use episode length as a diversity metric. As MAP-ELITES is the only non deep-RL based method, a manually generated policy was used. In the case of LunarLander, heuristics default controller included in the OpneAI’s package was used, and the agents was able to tweak the 8 parameters of the heuristics. In the case of the rest, a central pattern generator generating step function was used. Parameters controlling the pattern generator was 25 for Ant benchmark, and 14 for the BipedalWalker

ACQDRL

ACQDRL is our implementation of the Actor-Critic Quality Diversity Reinforcement Learning, a generalized framework for using MAP-Elites in Deep RL framework (68). Based on the said paper, we use the duty cycle, which is the ratio of time each leg spends in contact with the ground as the diversity metric for the BipedalWalker, and Ant. In the case of LunarLander, we use episode length as a diversity metric. Based on the discretization of policy space by the diversity metric, ACQDRL had 10 to 16 different behaviors included in its library.

4.4.3 Evaluation and Hyperparameters

For evaluation purposes, we selected 8 random scenarios from each benchmark. We evaluated each behavior 50 times against the selected scenario. By doing so, we marked the best-performing behavior for a given scenario. We then evaluated the best-performing behavior to its assigned scenario 100 times and reported the mean.

	Diversity Space Kernel, MAP-Elites and ACQDRL
BipedalWalker	2,3,5,10
LunarLander	2,3,5,10
Ant	2,3,5,10
	Scenario Space Kernel, CQD
BipedalWalker	2,3,5,10
LunarLander	2,3,5,10
Ant	2,3,5,10

Table 4.2: Hyperparameter Sweep for the SAC algorithm used in this chapter. Selected value in bold.

	BipedalWalker	LunarLander	Ant
Conditional	97.55±15.17	68.06±0.97	893.97±352.38
Manual	144.55±24.33	67.23±0.65	4373.87±405.22
Ensemble	109.32±13.95	54.13±2.44	4160.21±503.31
DIAYN	-14.63±0.59	58.35±1.12	35.93±62.94
MAP-ELITES	-87.12±40.36	9.23±2.82	-132.54±10.97
ACQDRL	-60.19±3.44	-126.19±4.72	1280.46±370.38
CQD (Ours)	180.70±10.53	69.28±0.08	4704.21±43.81

Table 4.3: Cumulative Performance of the Trained Algorithms

We used this approach as the scope of this chapter is on training a diverse and well-performing set of behaviors, not how we choose which behavior during test time.

We ran an evaluation at every 1e6 step. To balance between performance and computation cost, each baselines were tested with 5 random seeds. As the baselines require a base RL algorithm for policy optimization, we chose the SAC codes we used in the chapter 2. We trained a single-behavior version of the SAC to run a hyperparameter sweep. table 4.1 shows the hyperparameters swept and used for our SAC implementation.

For DIAYN, we used hyperparameters as listed in the original paper. For MAP-ELITES and ACQDRL, while we used hyperparameters proposed in their respective papers, we needed to tune the size of kernels for the diversity space. Likewise, our proposed approach has a hyperparameter controlling how many kernels each scenario axis is discretized into. table 4.2 shows the hyperparameters swept and how many kernels discretize each dimension.

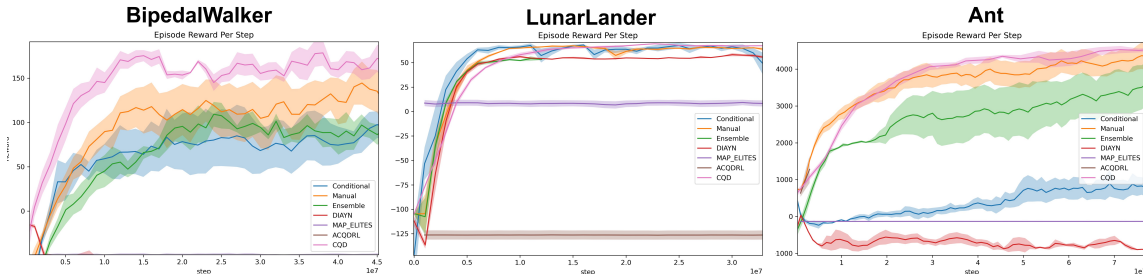


Figure 4.3: Training curve of the baselines across benchmarks. (From left to right) BipedalWalker, LunarLander, and Ant

4.5 Results

Figure 4.3 and Table 4.3 shows the training curves and the results of the trained library. A quick glance shows that our behavior outperforms all other baselines across all benchmarks.

One of the notable observations from the result is that while Conditional does well in LunarLander, it performs poorly in the BipedalWalker and the Ant benchmark. One of the differences between the LunarLander and BipedalWalker is the diversity and difference in modes of each behavior. In the case of LunarLander, the difference in behavior comes from how early the main engine should start firing to ensure a safe landing. The variations between behaviors are small enough to be supported by a single conditional network where multiple behaviors share network weights. However, in the case of the BipedalWalker and Ant, the gait pattern has to drastically change based on failure modes, such as walking as a stiff-kneed inverted pendulum, hopping on a leg or two, or having a natural-looking gait. For the environments where the difference in modality is greater, the Conditional benchmark tends to perform poorly.

Second, it should be noted that manual curriculum design, depending on the situation, can work quite well. However, as this approach is limited by expert domain knowledge, such an approach can miss better diversification and distribution of the behaviors, as shown in the case of BipedalWalker and LunarLander where our CQD clearly found a better library of behaviors.

Overall, DIAYN performs poorly across the benchmarks. While showing limited success in the LunarLander, DIAYN does not perform well in locomotion tasks where the modalities can differ so much that aiming for diversity often comes at a tradeoff of performance.

In the case of MAP-ELITES and ACQDRL, they perform poorly in general in our benchmarks. While MAP-Elites sometimes have a better performance at the start by using expert domain knowledge infused into the controller, both approaches do not learn diverse and well-performing behaviors that are useful. When MAP-ELITES and

ACQDRL train an agent, it collects its trajectory during training, uses the diversity metric to see which container in the library should the behavior be assigned to, and updates the container if the new behavior is performing better than the old one. However, in our setup where environmental parameters and dynamics vary during training, such evaluation on trajectory is difficult, as the resulting trajectory and the performance are not only dependent on the behavior but also on environmental factors and dynamics as well. Therefore, MAP-ELITES and ACQDRL will not work well if such external factors to performance cannot be removed for evaluating the quality and diversity of behavior.

4.6 Conclusion

In this chapter, we covered how curricular RL can be used in conjunction with quality diversity to generate a library of different and high-performing behaviors that can cumulatively achieve robustness. Using diversity within the scenario to ensure diversity in behavior, we achieve Quality-Diversity without manually designed diversity metric and outperform all baselines in the given benchmarks.

The limitation of our approach is that due to how we segmented the scenario space in grid-like cells, the 4X steps in the curriculum generation can be computationally costly. For future works, we could explore to expedite this process. Also, test time adaptation is a problem not covered in this chapter and could be covered in future works.

Chapter 5

Conclusion

In this paper, we have shown how curricular RL can be used to improve the robustness of the trained agents. In Chapter 2, we have shown how curriculum learning along with genetic algorithms can be used to improve the robustness of a trained agent in a single-agent domain. In Chapter 3, we explored how the idea can be expanded in the multi-agent domain, in conjunction with genetically optimized open-loop agents, can be used to achieve state-of-the-art performance in various multi-agent domains, both symmetrical and asymmetrical, competitive and collaborative. Finally, in Chapter 4, we have demonstrated how curricular learning can be used with quality-diversity to achieve robustness as a group. Overall, we have used curriculum learning to achieve all the stated goals with less expert supervision and in an environment-agnostic way, allowing the training to adapt to the environment.

I think one of the takeaways from the thesis research is that the theory only solves half of the problem. While theory can be used for verification of an algorithm and see if there's any theoretical limit that will prevent the algorithm from solving the problem, it is less efficient at predicting the actual performance of the algorithm as it is quite difficult to achieve the theoretical maximum. For example, most adversarial RL algorithms use a min-max formulation that guarantees robustness upon convergence, but due to the training instability triggered by the adversarial RL algorithm, it does not always work well. Likewise, we saw similar issues in multi-agent and other problem domains where the algorithm would not converge to its theoretical equilibrium within a reasonable time frame, especially due to training inefficiency and instability. Therefore, it would be wise to be careful about the underlying assumptions and environment-specific details when considering which algorithm to implement as especially in complex systems, the agent will be operating much lower than the maximum performance predicted by the theory.

Chapter 6

Appendix A. Additional results and implementation details for Chapter 3

6.1 Full Performance Results of the Trained Algorithms

In this section, we include the full results of our experiments. Table 6.1 and Table 6.2 shows the mean return of each algorithm against each other, while ?? and Table 6.3 includes the detailed results of each pair. Figure 6.1 shows the training curve, evaluated in round robin fashion.

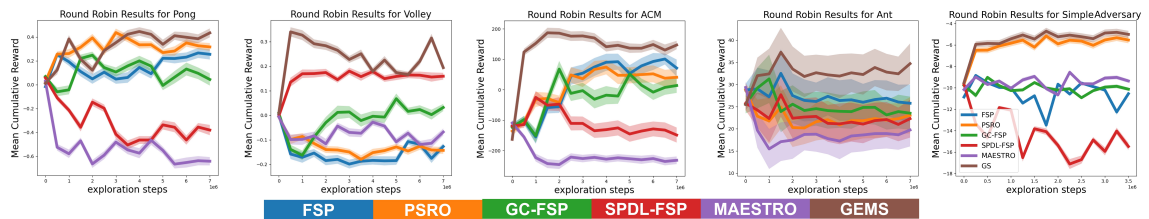


Figure 6.1: Training curve with round robin results.

6.2 Environment Details

This section will discuss the details of the benchmark environments used in Chapter 3.

	Pong	Volley	ACM
FSP	0.197 ± 0.158	-0.143 ± 0.061	94.498 ± 39.686
PSRO	0.141 ± 0.154	-0.145 ± 0.058	83.274 ± 51.499
GC+FSP	0.159 ± 0.179	-0.004 ± 0.07	-124.304 ± 44.356
SPDL+FSP	-0.386 ± 0.144	0.192 ± 0.058	-97.987 ± 51.969
MAESTRO	-0.578 ± 0.142	-0.094 ± 0.053	-226.845 ± 25.332
GEMS	0.466 ± 0.121	0.194 ± 0.058	139.080 ± 32.768

Table 6.1: Mean return of algorithms against baseline algorithms and ours for Pong, Volley, and ACM. The highest mean return in bold.

	Ant	SimpleAdversary
FSP	25.788 ± 4.228	-10.545 ± 0.166
PSRO	23.061 ± 4.066	-5.535 ± 0.308
GC+FSP	23.544 ± 3.735	-10.125 ± 0.158
SPDL+FSP	22.312 ± 4.275	-15.47 ± 0.338
MAESTRO	19.748 ± 3.706	-9.358 ± 0.169
GEMS	34.714 ± 4.549	-5.010 ± 0.333

Table 6.2: Mean return of algorithms against baseline algorithms and ours for Ant and SimpleAdversary. The highest mean return in bold.

Table 6.3: Detailed Win:Tie:Lose Ratio(%) of Each Algorithm Against Others.

Agent	vs FSP	vs PSRO	vs GC+FSP	vs SPDL+FSP	vs MAESTRO	vs GEMS (Ours)
Pong						
FSP	49±1 :2±0 :49±1					
PSRO	42±1 :1±0 :57±1	49±1 :3±0 :49±0				
GC+FSP	48±1 :1±0 :51±1	50±1 :0±0 :49±1	50±1 :0±0 :50±1			
SPDL+FSP	22±0 :0±0 :78±0	22±1 :1±0 :78±1	19±0 :0±0 :80±0	50±0 :0±0 :50±0		
MAESTRO	9±0 :0±0 :91±0	13±1 :0±0 :87±1	8±0 :0±0 :92±0	40±0 :0±0 :60±0	50±0 :0±0 :50±0	
GEMS (Ours)	66±1 :4±0 :30±0	63±1 :4±0 :33±1	73±1 :2±0 :25±1	89±0 :1±0 :11±0	94±0 :0±0 :6±0	47±1 :7±0 :47±1
Volley						
FSP	50±0 :0±0 :50±0					
PSRO	50±0 :0±0 :50±0	50±0 :0±0 :50±0				
GC+FSP	58±0 :0±0 :42±0	59±0 :0±0 :41±0	49±0 :2±0 :49±0			
SPDL+FSP	59±0 :16±0 :25±0	59±0 :15±0 :26±0	51±0 :21±0 :29±0	30±0 :41±0 :30±0		
MAESTRO	51±0 :0±0 :49±0	52±0 :0±0 :48±0	46±0 :0±0 :54±0	29±0 :15±0 :56±0	50±0 :0±0 :50±0	
GEMS (Ours)	61±0 :13±0 :26±0	58±0 :17±0 :25±0	52±0 :19±0 :29±0	31±0 :37±0 :31±0	54±0 :20±0 :27±0	31±0 :38±0 :31±0
ACM						
FSP	25±0 :49±0 :25±0					
PSRO	20±0 :54±0 :26±0	22±0 :55±0 :22±0				
GC+FSP	3±0 :48±0 :49±0	3±0 :50±0 :47±1	38±2 :23±1 :38±2			
SPDL+FSP	2±0 :48±0 :50±0	2±0 :49±0 :49±0	63±2 :23±1 :15±1	8±0 :84±0 :8±0		
MAESTRO	1±0 :39±0 :60±0	1±0 :39±0 :60±0	10±1 :13±1 :78±1	2±0 :5±0 :93±0	47±1 :6±0 :47±1	
GEMS (Ours)	32±0 :56±0 :14±0	35±0 :55±0 :12±0	50±0 :48±0 :2±0	48±0 :50±0 :1±0	62±0 :37±0 :1±0	18±0 :64±1 :18±0
Ant						
FSP	36±0 :27±0 :37±0					
PSRO	34±0 :23±0 :43±0	43±0 :15±0 :42±0				
GC+FSP	31±1 :27±1 :42±1	40±1 :20±1 :40±1	42±1 :20±1 :38±1			
SPDL+FSP	32±0 :22±0 :45±0	48±0 :17±0 :41±0	38±1 :20±0 :42±1	43±0 :16±0 :41±0		
MAESTRO	30±0 :25±1 :45±0	38±0 :18±0 :44±0	36±1 :23±1 :41±1	39±0 :18±0 :42±0	42±1 :18±0 :41±1	
GEMS (Ours)	36±0 :42±1 :22±1	51±1 :30±1 :19±1	47±1 :31±1 :22±1	49±1 :31±1 :20±1	51±0 :30±1 :18±1	26±1 :47±1 :27±1
SimpleAdversary						
FSP	50±0 :0±0 :50±0					
PSRO	53±0 :0±0 :47±0	50±0 :50±0 :50±0				
GC+FSP	50±0 :0±0 :50±0	46±0 :0±0 :54±0	50±0 :0±0 :50±0			
SPDL+FSP	46±0 :0±0 :54±0	43±0 :0±0 :57±0	47±0 :0±0 :53±0	50±0 :0±0 :50±0		
MAESTRO	49±0 :0±0 :51±0	47±0 :0±0 :53±0	49±0 :0±0 :51±0	52±0 :0±0 :48±0	50±0 :0±0 :50±0	
GEMS (Ours)	54±0 :0±0 :46±0	50±0 :0±0 :50±0	54±0 :0±0 :46±0	57±0 :0±0 :43±0	54±0 :0±0 :46±0	53±0 :0±0 :47±0

6.2.1 Pong

Environment Overview

Pong is a game similar to the one described in (13). In this game, there are two paddles, each located at the left and right edges of the map. Each agent controls a paddle capable of 1D movement. At the start of the game, a ball spawns at the center of the map with a certain velocity. The ball will bounce if it comes in contact with the top edge, the bottom edge, or paddles. An agent wins the game if the ball passes through the opposite edge of the map.

To play the game, each agent observes the position and velocity of the ball and the location of each paddle. The action space is 1-D. If the value is positive, the agent's paddle will move up at a constant speed, and vice versa if negative. An agent receives a reward of +1 if they win and -1 if they lose. If neither player manages to win the game after 300 steps, it is considered a tie with a reward of 0.

Environment Encoding

The original environment calls the random number generator four times to reset a game. Subsequently, we encode the environment with four values. One value determines whether the ego agent plays the left or right paddle. The other three control the initial velocity of the ball. One value determines whether the ball will be moving up or down, another controls the magnitude of the ball's velocity vector in the up-down axis, and the last one determines whether the ball will be traveling left or right. The magnitude of the ball's velocity in the left-right axis remains constant throughout the game.

6.2.2 Volley

Environment Overview

Volley is an environment based on the concept presented in (47), where two agents engage in a 2D volleyball game. This presents a more intricate update than Pong, as both players can move in a 2D space instead of the 1D movement of paddles. Additionally, the ball's horizontal speed can vary along with its vertical speed. At the start of the game, the ball spawns at the center of the map above the net, which divides the map into two halves. The ball will bounce upon contact with any players or edges except for the bottom. Each agent can redirect the ball by hitting it with the avatar they control. The ball follows a simple physics model, subject to gravity and simple elastic collision mechanics. An agent wins the game by successfully landing the ball on the other side of the map.

To play the game, each agent observes the position and velocity of the ball, the ego agent, and the other agent. Each agent has a 2-D continuous action space, dictating the desired vertical and horizontal velocity of the avatar in the game. While the avatar can move at the desired velocity in the horizontal axis, it can only be launched upward

with a desired velocity if it is on the ground. Otherwise, it will move along the surface or be in free fall due to gravity. An agent receives a reward of +1 if they win and -1 if they lose. If neither player manages to win the game after 300 steps, it is considered a tie, and both players receive a reward of 0.

Environment Encoding

The Volley environment is expressed in three values. The first two values are continuous (-1,1) and define the ball’s initial velocity. The third value is discrete 0,1 and defines whether the ego agent controls the avatar on the left or the right.

6.2.3 ACM

Environment Overview

ACM is a simulated dogfight environment. This is more complex than the previous two benchmarks, as ACM agents will move in 3D space with realistic physics simulation. At the start of the game, two aircraft spawn in the air at a certain orientation and a velocity vector with respect to the nose heading. The agent wins the game by pointing its nose at the opponent while avoiding getting pointed by the opponent’s nose. There is also a penalty involved with crashing to the ground.

To play the game, each agent observes the position, orientation, and velocity of the ego and the opponent agents. Each agent has a 4-D continuous action space with direct control over the aircraft’s elevator, aileron, rudder, and throttle. The physics is simulated by JSBSim (11), a high-fidelity simulator often used in autonomous aircraft research. The aerodynamics model for ACM is based on a Boeing F-15D, capable of flying two times faster than the speed of sound.

To win the game, the agent should have its nose pointed less than 5 degrees off the opponent aircraft while flying less than 2,000 feet away from the opponent. The agent can also win the game if the opponent aircraft flies below the hard deck of 500 ft. The agent wins the game if either of these conditions are met and loses if the opponent achieves either of the conditions. It is a tie if both aircraft achieve either of these conditions simultaneously. If either of the agents meets none of the conditions for 300 seconds, it is considered a tie. Agent receives a penalty of -300 for crashing into the ground, 150 for pointing the nose toward the target, and 100 if the opponent crashes to the ground. To help with training, there is a small, dense reward for getting closer to pointing the nose toward the opponent.

Environment Encoding

The ACM environment is expressed in 14 continuous values. They define the initial conditions of the game, which are each aircraft’s position, orientation, and airspeed.

6.2.4 Ant

Environment Overview

Ant is a simulated Mujoco environment where two ant agents try to flip or push the other agent out of the arena. With each agent having 8 degrees of freedom in a contact-rich setup, the environment offers a variety of behaviors to learn. For example, while ramming is often the only way to win against a stationary target, ramming is vulnerable to flipping attacks. However, flipping attacks are useless against stationary opponents as agents need more torque to lift and flip an opponent with all its legs firmly contacting the ground.

To play the game, each agent observes the position, orientation, velocity, and posture of the ego and the opponent agents. Each agent has 8-D continuous action space with direct control over the torques applied to each joint. The agents get a reward of 100 if the opponent is flipped or pushed out of the arena. There is a penalty of -20 if the agent is flipped or exits the arena. To make the training faster, there is a sparse reward for approaching the opponent or moving towards the center of the arena. If neither player manages to win the game after 300 steps, it is considered a tie, and both players receive a reward of 0. While the environment is based on OpenAI's implementation of Sumo-Ant (7), the arena is smaller to allow agents to interact more and learn faster.

Environment Encoding

The Ant environment is expressed in 1 discrete value assigning who will be controlling the red agent and who will be controlling the blue agent in the game. The red and blue agents have the same winning and losing conditions and have fixed spawn points in the arena.

6.2.5 SimpleAdversary

SimpleAdversary is a Multi Particle Environment from (110). SimpleAdversary provides an interesting opportunity to validate our GEMS on environments that are asymmetrical and have both competitive and collaborative features to a game. ?? shows a screenshot of the environment.

In this environment, there is one adversary agent (red), two good agents (blue), and two landmarks (green and black). One of the landmarks is a target landmark where the good agents get a reward for the good agents getting close to the agent and a negative reward for the adversary agent getting close to the target landmark. The vice versa applies to the reward for the adversary agent. While all agents can see the location of each other and landmarks, only good agents have additional information on where the goal is. The good agents observe the environment with 10-D space including the goal information while the adversary agent observes the environment

with 8-D space missing this environment. The action space is continuous, with the agents being allowed to move up, down, left, and right.

This environment covers challenges that are not covered well in the original manuscript. First, it is an asymmetrical environment where the goal and the optimal policies differ by role. Also, there are two good agents in the game which has to learn to collaborate with each other. Finally, there are three agents in this game, giving a limited yet first glimpse into scalability regarding the number of agents.

6.2.6 Environment Encoding

SimpleAdversary has a total of 5 objects in the game consisting of two landmarks, two good agents, and one adversary agent. The first 10 variables in the environment encoding cover the X and Y coordinates of the objects in the game. The last 11th variable in the environment encodes whether the ego agent is playing as the good agent or the adversary agent in the game.

As observation space is different whether playing as an adversary agent or a good agent, we added a zero padding to equalize the dimensions. In addition, we also add a variable to the observation space denoting whether the agent is playing as an adversary agent or a good agent. This allows the policy network to run as a conditional network able to support both roles in the game.

6.3 Implementation Details

This section covers the details of implementing the training procedures for each baseline for reproducibility purposes.

6.3.1 Training Hardware

We used an 88-core Intel Xeon Gold 6238 CPU at 2.10 GHz to train the models. Training models took from 5 ~ 15 days. Lighter benchmarks, like Pong, took 5 days, while heavier benchmarks, such as ACM, took 15 days. This is due to the computation load of the JSBsim, which provides high-fidelity aerodynamics simulation.

6.3.2 Tuning RL Algorithm

Tuning Procedures

To ensure that hyperparameters do not favor one baseline over another, they were tuned for the RL policy explorer in a single-agent setup against hand-coded opponents. For Pong and Volley, the opponents were PID controllers chasing the estimated impact point of the ball (75). In the case of ACM, a simple PID autopilot was employed to control the opponent aircraft’s speed, heading, and altitude. Ant was tuned with

Benchmark	Parameter Name	Range
Pong	Magnitude of the Ball's Vertical Speed	[0,1]
	Left-Right Initial Velocity Direction of the Ball	{0,1}
	Up-Down Initial Velocity Direction of the Ball	{0,1}
	Left / Right Paddle Controlled by the Ego Agent	{0,1}
Volley	Initial Horizontal Velocity of the Ball	[0,1]
	Initial Vertical Velocity of the Ball	[0,1]
	Left / Right Paddle Controlled by the Ego Agent	{0,1}
ACM	Ego Agent Initial X Coordinates	[-1,1]
	Ego Agent Initial Y Coordinates	[-1,1]
	Ego Agent Initial Z Coordinates	[-1,1]
	Ego Agent Initial Heading	[-1,1]
	Ego Agent Initial Pitch Angle	[-1,1]
	Ego Agent Initial Role Angle	[-1,1]
	Ego Agent Initial Airspeed	[-1,1]
	Opponent Agent Initial X Coordinates	[-1,1]
	Opponent Agent Initial Y Coordinates	[-1,1]
	Opponent Agent Initial Z Coordinates	[-1,1]
	Opponent Agent Initial Heading	[-1,1]
	Opponent Agent Initial Pitch Angle	[-1,1]
	Opponent Agent Initial Role Angle	[-1,1]
	Opponent Agent Initial Airspeed	[-1,1]
Ant	Ego and Opponent Agent Spawnpoint Configurations	{0,1}
SimpleAdversary	Good Agent 0 Location, X	[-1,1]
	Good Agent 0 Location, Y	[-1,1]
	Good Agent 1 Location, X	[-1,1]
	Good Agent 1 Location, Y	[-1,1]
	Adversary Agent 1 Location, X	[-1,1]
	Adversary Agent 1 Location, Y	[-1,1]
	Goal Location, X	[-1,1]
	Goal Location, Y	[-1,1]
	Obstacle Location, X	[-1,1]
	Obstacle Location, Y	[-1,1]
	Ego Agent is Playing As	{0,1}

Table 6.4: Summary of environmental parameters.

	Pong	Volley	ACM	Ant	SimpleAdversary
Start Steps	10000	10000	10000	N/A	10000
Learning Rate	3e-4	3e-4	3e-4	3e-4	3e-4
γ	0.98	0.98	0.98	0.98	0.98
α	auto	auto	auto	N/A	auto
Batch Size	256	256	512	36864	256
Replay Size	1e6	1e6	1e6	N/A	1e6
Update Every	1	1	1	36864	1
Backpropagation per Update	N/A	N/A	N/A	20	N/A

Table 6.5: Selected hyperparameters for RL algorithms.

a stationary agent that does not move. The combination of network structure and hyperparameters that performed the best in these single-agent setups was selected for our baseline experiments. Note that these hand-coded agents were only used to tune the RL hyperparameters. The hand-coded agents were not used when training the baselines and our approach, GEMS.

Network Structure

To train our RL algorithm, we experimented with various network structures, varying the depth of the hidden layers from 1 to 4 layers and the width from 64 to 512. Our findings revealed that the optimal architecture for Pong, Volley, and ACM consists of hidden layers composed of two fully connected layers, each with a size of 256. We utilized ReLU activation between the hidden layers and concluded that the last layer had an output dimension equivalent to the action dimension, followed by Tanh activation. Conversely, Ant uses 2-layer policy networks with Tanh activation and a width of 128.

6.3.3 RL Hyperparameters

We conducted a limited grid search to hyperparameters for our RL algorithm. For learning rate, we tried $\{3e-5, 1e-4, 3e-4, 1e-3\}$. For discount ratio γ , we tried $\{0.95, 0.98, 0.999\}$.

Regarding the algorithms that use SAC, we tried 0, 1000, 10000 for the initial exploration steps. For batch size, we tried $\{64, 256, 512\}$. For update every, we tried $\{1, 32, 100\}$. We tried $\{1e6, 3e6\}$ for Replay size. We ran each setting on 3 seeds to balance performance and computational cost.

Regarding algorithms that use PPO, we tried policy update frequency of 9216,18432,36864. For the number of times the gradient was calculated and backpropagated per update frequency, we tried 10,20,30. We ran each setting on 3 seeds to balance performance and computational cost.

Table 6.5 shows the selected hyperparameters we used for the main experiments.

	Pong	Volley	ACM	Ant
PSF				
Checkpoint Interval	10000	10000	10000	50000
PSRO				
Match Times	30	30	10	10
GC				
Evaluation Set Size	200	200	300	300
Curriculum Size	200	200	300	300
SPDL				
Penalty Proportion	1	1	0.1	0.1
Offset	20	20	20	20
MAESTRO				
Co-Player Exploration Coefficient	0.1	0.1	0.1	0.1
Curriculum Buffer Size	1000	1000	1000	1000

Table 6.6: Selected hyperparameters for baselines.

6.4 Baseline Hyperparameter Tuning

To tune the hyperparameters for the baselines, we trained the models using the following settings. Since it is challenging to tune the hyperparameters by conducting a round-robin across all hyperparameter settings used for different baselines, each setting was evaluated against the same hand-crafted opponents used for tuning the RL hyperparameters. Striking a balance between performance and computational cost, we ran each configuration with 3 seeds. Table 6.6 presents the selected hyperparameters used for the main experiments.

We tested the following hyperparameter values in a limited grid search. For adding checkpoints, we tried saving a copy of the agents’ policies at every $\{10000, 30000, 50000\}$ exploration step. We used the same interval of steps as the interval in which curriculum is generated for the algorithms that generate one at every epoch. For measuring performance between each checkpoint on random environment parameters to approximate the Nash equilibrium in PSRO, we tried $\{10,30\}$ evaluations per pair. As for the hyperparameters of the GC, we tried $\{200,300\}$ for the size of the population of scenarios evaluated for whether being solved or not solved. We also tried sizes of $\{200,300\}$ for the sizes of the generated curriculum and mutation rates of $\{0.1,1\}$. For SPDL, we tried values of $\{0,10,20\}$ for offset and $\{0.1,0.3,1\}$ for penalty proportion. For MAESTRO, we tried co-player exploration coefficients of $\{0.05,0.1\}$ and curriculum buffer sizes of $\{500,1000\}$. For GEMS, we copied the hyperparameters from the GC in terms of the size of the curriculum and mutation rate. This was done to make comparisons between different algorithms easier by removing the effect of different hyperparameters during genetic operations.

6.5 Crossover and Mutation

This section covers how GEMS perform crossover and mutation. Please note that these steps simply conduct generic genetic operations and are not hand-tuned nor hardcoded to specifically solve the benchmark tasks in this paper.

For a sampled parent scenarios \mathbf{m}, \mathbf{n} , the corresponding encoding would look like;

$$\mathbf{m} = \{i_\pi, \pi_\emptyset, \psi\} = \{m_0, m_1\} = \{m_{0,0}, m_{0,1} \dots m_{0,x}, m_{1,0}, m_{1,1} \dots m_{1,y}\} \quad (6.1)$$

$$\mathbf{n} = \{i_\pi, \pi_\emptyset, \psi\} = \{n_0, n_1\} = \{n_{0,0}, n_{0,1} \dots n_{0,x}, n_{1,0}, n_{1,1} \dots n_{1,z}\} \quad (6.2)$$

\mathbf{m}_o encodes the choice of opponent to load i_π and the environment parameters ψ while \mathbf{m}_e represents the encoding of the GenOpt Agent π_\emptyset . In this section, x corresponds to the length of environment encoding for \mathbf{m} . y and z in this section corresponds to the length of GenOpt Agent π_\emptyset encoding for \mathbf{m} and \mathbf{n} . If using the same environment, x will not be different between the scenarios, whereas y and z will vary depending on the encoded environment.

GEMS performs crossover by swamping a section from one parent with a section from another parent. This would be done by randomly selecting three split points in the encoding. Using a uniform distribution, GEMS sample three integer values $\eta_0 \in \{0, 1, 2, \dots, x\}, \eta_1 \in \{0, 1, 2, \dots, y\}, \eta_2 \in \{0, 1, 2, \dots, z\}$

From this, GEMS generates two child scenarios \mathbf{p}, \mathbf{q} as follows;

$$\mathbf{p} = \{m_{0,0} \dots m_{0,\eta_0}, n_{0,\eta_0+1} \dots n_{0,x}, m_{1,0} \dots m_{1,\eta_1}, n_{1,\eta_1+1} \dots n_{1,z}\} \quad (6.3)$$

$$\mathbf{q} = \{n_{0,0} \dots n_{0,\eta_0}, m_{0,\eta_0+1} \dots m_{0,x}, n_{1,0} \dots n_{1,\eta_2}, m_{1,\eta_2+1} \dots n_{1,y}\} \quad (6.4)$$

For the special cases such as $\eta_0 = 0, x$ or $\eta_1 = 0, y$, or $\eta_2 = 0, z$, GEMS inherits the segment without dividing it. For example, if $\eta_0 = 0$, then $\mathbf{p}_0 = \mathbf{n}_0, \mathbf{q}_0 = \mathbf{m}_0$ and so on.

To perform a mutation on a sampled scenario, GEMS first performs a crossover between the scenario and a randomly generated scenario and returns one of the child scenarios as the mutated scenario. The mutation rate controls the probability of each scenario undergoing mutation.

6.6 Sampling Parents

In this paper, we have defined the probability of GEMS sampling scenarios as parents as the following;

$$p(\xi) \propto \delta(\xi)(1 - G(\pi_{ego}, \xi)) \quad (6.5)$$

In practice, GEMS uses the normalized form as follows;

$$p(\xi_i) = \frac{\delta(\xi_i)(1 - G(\pi_{ego}, \xi_i))}{\sum \delta(\xi)(1 - G(\pi_{ego}, \xi))} \quad (6.6)$$

If $\sum \delta(\xi) = 0$, GEMS use regret-only version as follows;

$$p(\xi_i) = \frac{(1 - G(\pi_{ego}, \xi_i))}{\sum (1 - G(\pi_{ego}, \xi))} \quad (6.7)$$

6.7 Evolution of Curriculum

Figure 6.2 includes characteristic examples of the scenarios generated and used as curriculum for each algorithm in the ACM benchmark.

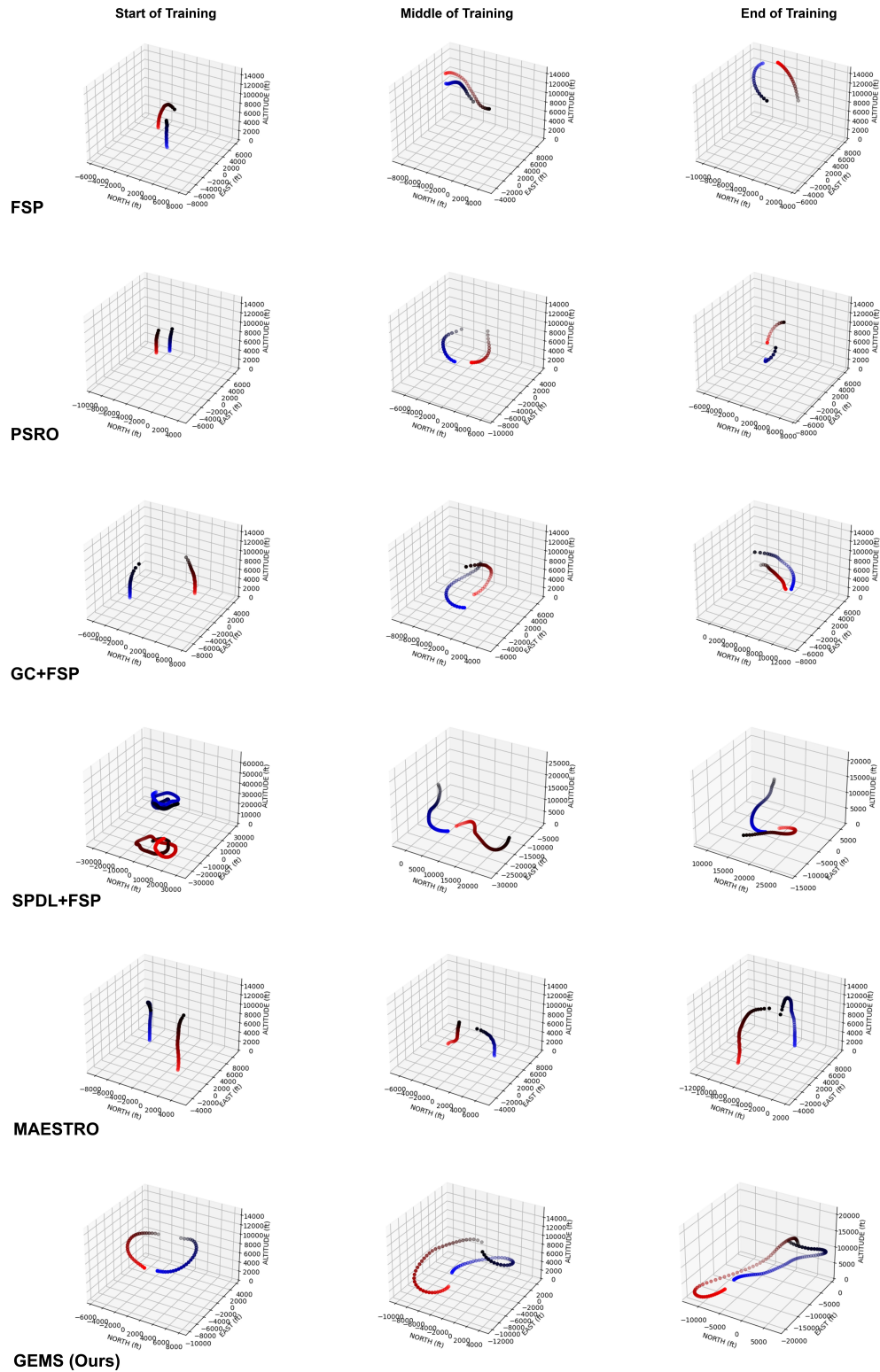


Figure 6.2: Each dot marks the position of the red and blue aircraft at 1-second intervals. The color of the markers transitions from black to blue for the student aircraft and from black to red for the opponent aircraft. At the start of the training, only our GEMS can present scenarios and opponents that provide interesting data points instead of simply crashing to the ground. While SPDL+FSP somewhat succeeds in finding scenarios that do not end in a crash, it optimizes scenarios without considering opponent policy. It simply finds trivial cases where agents are flying in circles.

Bibliography

- [1] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik’s cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.
- [2] Stefano V Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018.
- [3] Aaron D Ames, Samuel Coogan, Magnus Egerstedt, Gennaro Notomista, Koushil Sreenath, and Paulo Tabuada. Control barrier functions: Theory and applications. In *2019 18th European control conference (ECC)*, pages 3420–3431. IEEE, 2019.
- [4] Minoru Asada, Shoichi Noda, Sukoya Tawaratsumida, and Koh Hosoda. Purposive behavior acquisition for a real robot by vision-based reinforcement learning. *Machine learning*, 23(2-3):279–303, 1996.
- [5] David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech Czarnecki, Julien Perolat, Max Jaderberg, and Thore Graepel. Open-ended learning in symmetric zero-sum games. In *International Conference on Machine Learning*, pages 434–443. PMLR, 2019.
- [6] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2017.
- [7] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. Emergent complexity via multi-agent competition. In *International Conference on Learning Representations*, 2018.
- [8] Nikhil Barhate. Minimal pytorch implementation of proximal policy optimization. <https://github.com/nikhilbarhate99/PP0-PyTorch>, 2021.

- [9] Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1171–1179, 2015.
- [10] Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.
- [11] Jon Berndt. Jsbsim: An open source flight dynamics model in c++. In *AIAA Modeling and Simulation Technologies Conference and Exhibit*, page 4923, 2004.
- [12] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.
- [13] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.
- [14] George W Brown. Iterative solution of games by fictitious play. *Act. Anal. Prod Allocation*, 13(1):374, 1951.
- [15] George H Burgin and LB Sidor. Rule-based air combat simulation. Technical report, 1988.
- [16] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.
- [17] Dian Chen, Vladlen Koltun, and Philipp Krähenbühl. Learning to drive from a world on rails. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15590–15599, 2021.
- [18] Kamil Andrzej Ciosek and Shimon Whiteson. Offer: Off-environment reinforcement learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.
- [19] createmind. Deep reinforcement learning. <https://github.com/createamind/DRL/tree/master/spinup/envs/BipedalWalkerHardcore>, 2019.
- [20] Antoine Cully, Jeff Clune, and Jean-Baptiste Mouret. Robots that can adapt like natural animals. *arXiv preprint arXiv:1407.3501*, 2, 2014.
- [21] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521(7553):503–507, 2015.

- [22] Felipe Leno Da Silva and Anna Helena Reali Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64:645–703, 2019.
- [23] Charles Darwin. *On the origin of species: A facsimile of the first edition*. Harvard University Press, 1964.
- [24] Jonas Degraeve, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature*, 602(7897):414–419, 2022.
- [25] Michael Dennis, Natasha Jaques, Eugene Vinitzky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.
- [26] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.
- [27] Yuqing Du, Pieter Abbeel, and Aditya Grover. It takes four to tango: Multiagent selfplay for automatic curriculum generation. *arXiv preprint arXiv:2202.10608*, 2022.
- [28] Yan Duan, John Schulman, Xi Chen, Peter L Bartlett, Ilya Sutskever, and Pieter Abbeel. RL²: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.
- [29] Benjamin Ellenberger. Pybullet gymperium. <https://github.com/benelot/pybullet-gym>, 2018–2019.
- [30] Yousef Emam, Gennaro Notomista, Paul Glotfelter, Zsolt Kira, and Magnus Egerstedt. Safe reinforcement learning using robust control barrier functions. *IEEE Robotics and Automation Letters*, 2022.
- [31] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.
- [32] Shuo Feng, Haowei Sun, Xintao Yan, Haojie Zhu, Zhengxia Zou, Shengyin Shen, and Henry X Liu. Dense reinforcement learning for safety validation of autonomous vehicles. *Nature*, 615(7953):620–627, 2023.
- [33] Xidong Feng, Oliver Slumbers, Ziyu Wan, Bo Liu, Stephen McAleer, Ying Wen, Jun Wang, and Yaodong Yang. Neural auto-curricula in two-player zero-sum games. *Advances in Neural Information Processing Systems*, 34:3504–3517, 2021.

- [34] Manon Flageat, Felix Chalumeau, and Antoine Cully. Empirical analysis of pga-map-elites for neuroevolution in uncertain domains. *ACM Transactions on Evolutionary Learning*, 3(1):1–32, 2023.
- [35] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pages 1515–1528. PMLR, 2018.
- [36] Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Conference on robot learning*, pages 482–495. PMLR, 2017.
- [37] Pierre Fournier, Mohamed Chetouani, Pierre-Yves Oudeyer, and Olivier Sigaud. Accuracy-based curriculum learning in deep reinforcement learning.
- [38] Marta Garnelo, Wojciech Marian Czarnecki, Siqi Liu, Dhruva Tirumala, Junhyuk Oh, Gauthier Gidel, Hado van Hasselt, and David Balduzzi. Pick your battles: Interaction graphs as population-level objectives for strategic diversity. *arXiv preprint arXiv:2110.04041*, 2021.
- [39] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. *arXiv preprint arXiv:1905.10615*, 2019.
- [40] Jack H Good, Nicholas Gisolfi, Kyle Miller, and Artur Dubrawski. Verification of fuzzy decision trees. *IEEE Transactions on Software Engineering*, 49(5):3277–3288, 2023.
- [41] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [42] Mononito Goswami, Cristian Challu, Laurent Callot, Lenon Minorics, and Andrey Kan. Unsupervised model selection for time-series anomaly detection. *arXiv preprint arXiv:2210.01078*, 2022.
- [43] Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1311–1320. JMLR. org, 2017.
- [44] Wenbo Guo, Xian Wu, Sui Huang, and Xinyu Xing. Adversarial policy learning in two-player competitive games. In *International Conference on Machine Learning*, pages 3910–3919. PMLR, 2021.

- [45] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. 2018.
- [46] Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. In *International Conference on Learning Representations*, 2020.
- [47] Hardmaru. Slimevolleygym. <https://github.com/hardmaru/slimevolleygym>, 2020.
- [48] Tairan He, Chong Zhang, Wenli Xiao, Guanqi He, Changliu Liu, and Guanya Shi. Agile but safe: Learning collision-free high-speed legged locomotion. *arXiv preprint arXiv:2401.17583*, 2024.
- [49] Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *International conference on machine learning*, pages 805–813. PMLR, 2015.
- [50] Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. 2016.
- [51] David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. 2018.
- [52] David Howard, Humphrey Munn, Davide Dolcetti, Josh Kannemeyer, and Nicole Robinson. Assessing evolutionary terrain generation methods for curriculum reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 377–384, 2022.
- [53] Boris Ivanovic, James Harrison, Apoorva Sharma, Mo Chen, and Marco Pavone. Barc: Backward reachability curriculum for robotic reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 15–21. IEEE, 2019.
- [54] Garud N Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 30(2):257–280, 2005.
- [55] Jean-Baptiste Jeannin, Khalil Ghorbal, Yanni Kouskoulas, Ryan Gardner, Aurora Schmidt, Erik Zawadzki, and André Platzer. Formal verification of acas x, an industrial airborne collision avoidance system. In *2015 international conference on embedded software (EMSOFT)*, pages 127–136. IEEE, 2015.
- [56] Lu Jiang, Deyu Meng, Qian Zhao, Shiguang Shan, and Alexander G Hauptmann. Self-paced curriculum learning. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

- [57] Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. *Advances in Neural Information Processing Systems*, 34:1884–1897, 2021.
- [58] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR, 2021.
- [59] Andrej Karpathy and Michiel Van De Panne. Curriculum learning for motor skills. In *Canadian Conference on Artificial Intelligence*, pages 325–330. Springer, 2012.
- [60] Rituraj Kaushik, Timothée Anne, and Jean-Baptiste 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)*, pages 5269–5276. IEEE, 2020.
- [61] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [62] Pascal Klink, Carlo D’Eramo, Jan Peters, and Joni Pajarinen. Self-paced deep reinforcement learning. In *NeurIPS*, 2020.
- [63] M Pawan Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems*, pages 1189–1197, 2010.
- [64] Alexey Kurakin, Ian Goodfellow, Samy Bengio, Yinpeng Dong, Fangzhou Liao, Ming Liang, Tianyu Pang, Jun Zhu, Xiaolin Hu, Cihang Xie, et al. Adversarial attacks and defences competition. In *The NIPS’17 Competition: Building Intelligent Systems*, pages 195–231. Springer, 2018.
- [65] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30, 2017.
- [66] David S Leslie and Edmund J Collins. Generalised weakened fictitious play. *Games and Economic Behavior*, 56(2):285–298, 2006.
- [67] Yongyuan Liang, Yanchao Sun, Ruijie Zheng, and Furong Huang. Efficient adversarial training without attacking: Worst-case-aware robust reinforcement learning. *Advances in Neural Information Processing Systems*, 35:22547–22561, 2022.

- [68] Bryan Lim, Manon Flageat, and Antoine Cully. Understanding the synergies between quality-diversity and deep reinforcement learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1212–1220, 2023.
- [69] Qian Long, Zihan Zhou, Abhibav Gupta, Fei Fang, Yi Wu, and Xiaolong Wang. Evolutionary population curriculum for scaling multi-agent reinforcement learning. *arXiv preprint arXiv:2003.10423*, 2020.
- [70] Shie Mannor, Ofir Mebel, and Huan Xu. Lightning does not strike twice: robust mdps with coupled uncertainty. In *Proceedings of the 29th International Conference on Machine Learning*, pages 451–458, 2012.
- [71] Stephen McAleer, John B Lanier, Roy Fox, and Pierre Baldi. Pipeline psro: A scalable approach for finding approximate nash equilibria in large games. *Advances in neural information processing systems*, 33:20238–20248, 2020.
- [72] Stephen McAleer, John B Lanier, Kevin A Wang, Pierre Baldi, and Roy Fox. Xdo: A double oracle algorithm for extensive-form games. *Advances in Neural Information Processing Systems*, 34:23128–23139, 2021.
- [73] Stephen McAleer, Kevin Wang, John B Lanier, Marc Lanctot, Pierre Baldi, Tuomas Sandholm, and Roy Fox. Anytime psro for two-player zero-sum games. 2022.
- [74] Viraj Mehta, Vikramjeet Das, Ojash Neopane, Yijia Dai, Ilija Bogunovic, Jeff Schneider, and Willie Neiswanger. Sample efficient reinforcement learning from human feedback via active exploration.
- [75] Nicolas Minorsky. Directional stability of automatically steered bodies. *Journal of the American Society for Naval Engineers*, 34(2):280–309, 1922.
- [76] Sanmit Narvekar and Peter Stone. Learning curriculum policies for reinforcement learning. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 25–33. International Foundation for Autonomous Agents and Multiagent Systems, 2019.
- [77] John F Nash Jr. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.
- [78] Arnab Nilim and Laurent El Ghaoui. Robust control of markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.
- [79] OpenAI. Copilot - your ai pair programmer. <https://github.com/features/copilot>, 2022.

- [80] OpenAI. Introducing chatgpt. <https://openai.com/blog/chatgpt>, 2023.
- [81] Gregory Palmer, Chris Parry, Daniel JB Harrold, and Chris Willis. Deep reinforcement learning for autonomous cyber operations: A survey. *arXiv preprint arXiv:2310.07745*, 2023.
- [82] Xinlei Pan, Daniel Seita, Yang Gao, and John Canny. Risk averse robust adversarial reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8522–8528. IEEE, 2019.
- [83] John Paparrizos, Yuhao Kang, Paul Boniol, Ruey S Tsay, Themis Palpanas, and Michael J Franklin. Tsb-uad: an end-to-end benchmark suite for univariate time-series anomaly detection. *Proceedings of the VLDB Endowment*, 15(8):1697–1711, 2022.
- [84] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pages 17473–17498. PMLR, 2022.
- [85] Abhishek Paudel, Xuesu Xiao, and Gregory J Stein. Multi-strategy deployment-time learning and adaptation for navigation under uncertainty. In *8th Annual Conference on Robot Learning*.
- [86] Supratik Paul, Konstantinos Chatzilygeroudis, Kamil Ciosek, Jean-Baptiste Mouret, Michael A Osborne, and Shimon Whiteson. Alternating optimisation and quadrature for robust control. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.
- [87] Supratik Paul, Michael A Osborne, and Shimon Whiteson. Fingerprint policy optimisation for robust reinforcement learning. In *International Conference on Machine Learning*, pages 5082–5091. PMLR, 2019.
- [88] Supratik Paul, Michael A Osborne, and Shimon Whiteson. Fingerprint policy optimisation for robust reinforcement learning. In *International Conference on Machine Learning*, pages 5082–5091. PMLR, 2019.
- [89] Thomas Pierrot, Valentin Macé, Felix Chalumeau, Arthur Flajolet, Geoffrey Cideron, Karim Beguir, Antoine Cully, Olivier Sigaud, and Nicolas Perrin-Gilbert. Diversity policy gradient for sample efficient quality-diversity optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1075–1083, 2022.
- [90] Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2817–2826. JMLR. org, 2017.

- [91] Adrian P Pope, Jaime S Ide, Daria Mićović, Henry Diaz, David Rosenbluth, Lee Ritholtz, Jason C Twedt, Thayne T Walker, Kevin Alcedo, and Daniel Javorsek. Hierarchical reinforcement learning for air-to-air combat. In *2021 international conference on unmanned aircraft systems (ICUAS)*, pages 275–284. IEEE, 2021.
- [92] Adrian P Pope, Jaime S Ide, Daria Mićović, Henry Diaz, Jason C Twedt, Kevin Alcedo, Thayne T Walker, David Rosenbluth, Lee Ritholtz, and Daniel Javorsek. Hierarchical reinforcement learning for air combat at darpa’s alphasdogfight trials. *IEEE Transactions on Artificial Intelligence*, 2022.
- [93] Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *Conference on Robot Learning*, pages 835–853. PMLR, 2020.
- [94] Antonin Raffin, Ashley Hill, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, and Noah Dormann. Stable baselines3. <https://github.com/DLR-RM/stable-baselines3>, 2019.
- [95] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pages 5331–5340. PMLR, 2019.
- [96] Laminar Research. X-plane 11. <https://www.x-plane.com>, 2017.
- [97] Leslie Rice, Eric Wong, and Zico Kolter. Overfitting in adversarially robust deep learning. In *International Conference on Machine Learning*, pages 8093–8104. PMLR, 2020.
- [98] Pouya Samangouei, Maya Kabkab, and Rama Chellappa. Defense-gan: Protecting classifiers against adversarial attacks using generative models. In *International Conference on Learning Representations*, 2018.
- [99] Mikayel Samvelyan, Akbir Khan, Michael D Dennis, Minqi Jiang, Jack Parker-Holder, Jakob Nicolaus Foerster, Roberta Raileanu, and Tim Rocktäschel. Maestro: Open-ended environment design for multi-agent reinforcement learning. In *The Eleventh International Conference on Learning Representations*.
- [100] Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: a comprehensive evaluation. *Proceedings of the VLDB Endowment*, 15(9):1779–1797, 2022.
- [101] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

- [102] Jaemin Seo, SangKyeun Kim, Azarakhsh Jalalvand, Rory Conlin, Andrew Rothstein, Joseph Abbate, Keith Erickson, Josiah Wai, Ricardo Shousha, and Egemen Kolemen. Avoiding fusion plasma tearing instability with deep reinforcement learning. *Nature*, 626(8000):746–751, 2024.
- [103] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.
- [104] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.
- [105] Max Olan Smith, Thomas Anthony, Yongzhao Wang, and Michael P Wellman. Learning to play against any mixture of opponents. *arXiv preprint arXiv:2009.14180*, 2020.
- [106] Max Olan Smith, Thomas Anthony, and Michael P Wellman. Iterative empirical game solving via single policy best response. *arXiv preprint arXiv:2106.01901*, 2021.
- [107] Yeeho Song and Jeff Schneider. Genetic algorithm for curriculum design in multi-agent reinforcement learning. In *8th Annual Conference on Robot Learning*.
- [108] Yeeho Song and Jeff Schneider. Robust reinforcement learning via genetic curriculum. In *2022 International Conference on Robotics and Automation (ICRA)*, 2022.
- [109] Adaptive Agent Team, Jakob Bauer, Kate Baumli, Satinder Baveja, Feryal Behbahani, Avishkar Bhoopchand, Nathalie Bradley-Schmieg, Michael Chang, Natalie Clay, Adrian Collister, et al. Human-timescale adaptation in an open-ended task space. *arXiv preprint arXiv:2301.07608*, 2023.
- [110] J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.
- [111] Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.
- [112] Chen Tessler, Yonathan Efroni, and Shie Mannor. Action robust reinforcement learning and applications in continuous control. In *International Conference on Machine Learning*, pages 6215–6224. PMLR, 2019.

- [113] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE, 2012.
- [114] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.
- [115] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Poet: open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 142–151, 2019.
- [116] Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeffrey Clune, and Kenneth Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *International Conference on Machine Learning*, pages 9940–9951. PMLR, 2020.
- [117] Jungdam Won, Deepak Gopinath, and Jessica Hodgins. Control strategies for physically simulated characters performing two-player competitive sports. *ACM Transactions on Graphics (TOG)*, 40(4):1–11, 2021.
- [118] Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.
- [119] Chaowei Xiao, Bo Li, Jun-Yan Zhu, Warren He, Mingyan Liu, and Dawn Song. Generating adversarial examples with adversarial networks. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, pages 3905–3911, 2018.
- [120] Chaojian Yu, Bo Han, Li Shen, Jun Yu, Chen Gong, Mingming Gong, and Tongliang Liu. Understanding robust overfitting of adversarial training and beyond. In *International Conference on Machine Learning*, pages 25595–25610. PMLR, 2022.
- [121] Wojciech Zaremba and Ilya Sutskever. Learning to execute. 2015.