The enemy of my enemy is my friend: Learning to Generate Goals via Curriculum Self Play

Anonymous Author(s)
Affiliation
Address
email

Abstract

We are interested in training general-purpose reinforcement learning agents that can solve a wide variety of goals. Training such agents without excessive interactions requires automatic generation of a *goal curriculum*. This is challenging as it requires (a) exploring goals of increasing difficulty while ensuring that the agent (b) is exposed to a diverse set of goals in a sample efficient manner and (c) does not catastrophically forget previously solved goals. We propose *Curriculum Self Play* (CuSP), an automated goal generation framework that seeks to satisfy these desiderata by virtue of a multi-player game with 4 agents. Our method builds on the two-player method introduced in Asymmetric Self-Play and carefully balances cooperation and competition between two off-policy student learners and two regret-maximizing teachers that generate goals which are challenging yet feasible. The teachers also account for goal coverage and the non-stationary nature of the students, allowing us to automatically induce a curriculum that balances progressive exploration with anti-catastrophic exploitation. Our experiments demonstrate that our method succeeds at generating effective curricula of goals for a range of complex robotics tasks and significantly outperforms ASP and other competing methods at zero-shot test-time generalization to novel out-of-distribution goals.

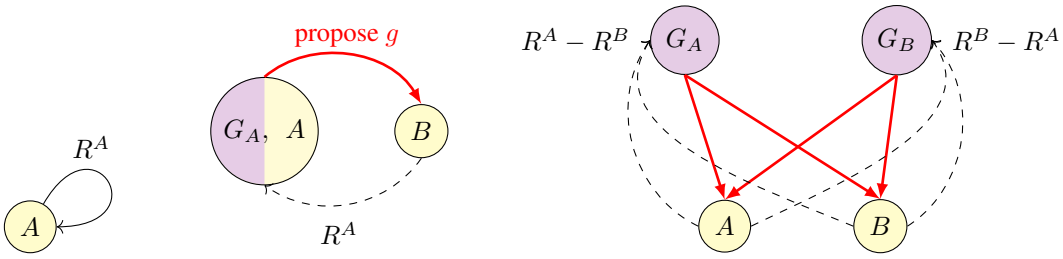
1 Introduction

Reinforcement learning (RL) has seen great success in a range of applications, from game-playing [37, 33] to real world robotics tasks [27]. However, these accomplishments are often in the realm of mastering a single specific task. For many applications, we further desire agents who are capable of generalizing to a variety of tasks. This can achieved via goal-conditioned RL, where an RL agent is trained to solve many goals by specifying a sequence of goals, i.e. a goal curriculum. Manually designing such curricula is tedious and moreover, does not scale due to being environment specific and heuristic-driven. Even when it is easy to engineer a curriculum, we would prefer curricula that can dynamically adapt to the learner's current capabilities. For example, effective human tutors use an understanding of their pupil's current knowledge and learning trajectory to create a curriculum.

Our work builds on two broad desiderata for curriculum generation [4] that enable sample-efficient learning of goal-conditioned policies: *progressive exploration* and *anti-catastrophic exploitation*. Progressive exploration refers to curricula where the agent gradually learns to solve tasks of increasing difficulty and diversity. Anti-catastrophic exploitation ensures that the agent periodically resolves previously attempted goals that could become harder to solve during the course of training a.k.a. catastrophic forgetting [9, 17].

Many prior works have attempted to incorporate aspects of these criteria for automatic curriculum generation. One approach is to parameterize a goal generator and pit it against a discriminator that assesses goal difficulty [8] or feasibility [32] for the current agent. Racaniere et al. [32] also address

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.



(a) Self Play (b) Asymmetric Self Play (c) Curriculum Self Play

Figure 1: Interactions between learners A , B and teachers G_A, G_B in Self Play variations. In (a), A plays against variations of itself in a zero-sum game to automatically generate a curriculum. In (b), A acts as both an agent and a demonstrator by being rewarded for proposing hard goals to B. In (c), we separate the goal generation from the demonstrators and symmetrize the whole system.

desired qualities of validity and diversity by crafting specific losses for these characteristics. However, these approaches rely on learning an accurate discriminator for assessing goal difficulty, which can be challenging. Rather than employing discriminators in the standard adversarial minimax formulation, Dennis et al. [6] proposes an alternate minimax objective based on the agent's regret. The 'expert' used for computing the regret is another goal-conditioned policy whose utility is maximized by an environment generator.

An alternative method for automatic curriculum generation that has seen great success is self-play, where an agent plays against other versions of itself [37, 3, 2] (Fig. 1a). Unfortunately, this method does not apply directly to complex tasks where we do not have the structure of a two player, zero-sum game. A promising variation is Asymmetric Self-Play (ASP) [40], which involves a minimax game between two similar agents: a goal demonstrator, Alice, and the desired goal-conditioned agent, Bob (Fig. 1b). Bob is asked to solve or reverse tasks demonstrated by Alice in the same environment. To motivate Alice to propose successively challenging goals, Alice is rewarded whenever Bob fails. This method inherently captures an easy-to-hard curriculum, since any goal solved by Alice is achievable by Bob in principle. However, ASP tends to be sample inefficient as it is bottlenecked by Alice's ability to remember earlier goals and explore a diverse goal space.

Our work aims to bring together the strengths of goal generative methods and ASP while addressing the two desiderata. Our work focuses on the multi-goal setting with the curriculum generator controlling goals [31]. We propose *Curriculum Self Play* (CuSP), a symmetric multi-agent setup for automatic curricula generation. CuSP optimizes for two off-policy goal-conditioned students, Alice and Bob, and two regret-maximizing goal generators, G_A and G_B (Fig. 1c). Drawing inspiration from the minimax regret objective in PAIRED [6], we designate each goal generator as being 'friendly' to one of the students (G_A for Alice, G_B for Bob), and optimize each goal generator using the difference in utility between the preferred student and the other student (i.e., G_A is rewarded $R^A - R^B$, where R^A, R^B are Alice's and Bob's returns respectively when challenged with the same goal). This encourages proposing goals that are neither too easy nor too hard. With our symmetrized setup, Alice and Bob each have a 'friendly' goal generator that is inclined to propose more feasible goals without being too simple, and an 'unfriendly' goal generator that is inclined to propose more challenging goals. As the tasks increase in difficulty overall, the former's cooperation helps in scaffolding learning while the latter's competition helps for pushing the boundaries of the agent's capabilities.

We further address goal diversity and sample efficiency by leveraging the off policy entropy-regularized optimization of Soft Actor Critic (SAC) [13] for training the goal generators. The goal generators' replay buffers track the regrets for all proposed goals. To account for non-stationary learners, we propose an update rule for the regrets in the goal generators' replay buffers based on the critic networks of Alice and Bob. In doing so, we constantly keep track of the learners' current abilities and can propose increasingly challenging goals. At the same time, using the regret objective also places the current abilities in context. For example, if Bob has forgotten how to achieve a particular goal, the regret $R^A - R^B$ increases and G_A is more inclined to re-propose that goal. Thus the combination of using a replay buffer and the regret objective aids in addressing the twin goals of progressive exploration and anti-catastrophic exploitation.

Compared to prior methods for goal generation, we demonstrate that CuSP is able to satisfy the desiderata for good curricula. We show that this translates to improved sample efficiency and

generalization to novel out-of-distribution goals across a range of robotics tasks, encompassing navigation, manipulation, and locomotion. We highlight some emergent skills that arise via CuSP when generalizing to particularly challenging goals, and find that our method is able to handle cases when portions of the goal space are infeasible.

2 Background

Problem Setup. We model the environment and the goal-conditioned learner using a Markov decision process, $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{G}, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the environment transition function, $\mathcal{R} : \mathcal{G} \times \mathcal{S} \to \mathbb{R}$ is the goal-conditioned reward function, $\mathcal{G} \subseteq \mathcal{S}$ is the goal space, and γ is a discount factor. We consider both dense and sparse reward functions: $r(s|g) = -d(s, g)$, where d is a distance metric (e.g. L_2 norm) between the state s and the goal g for the former, or $r(s|g) = \mathbb{1}\{d(s, g) < \epsilon\}$ for the latter. Our objective is to learn a policy $\pi(a_t|s_t, g)$ that maximizes the success rate of reaching all goals $g \in \mathcal{G}$. To assess ability to generalize to out-of-distribution goals, we split the goal space into $\mathcal{G}_{id}$ and $\mathcal{G}_{ood}$ where the goal generators can only propose goals $g \in \mathcal{G}_{id}$, and we evaluate on sampled $g \in \mathcal{G}_{ood}$.

Self Play (SP). SP is an approach for automatic curriculum generation in competitive, zero-sum games. In this paradigm, an agent plays against past versions of itself in order to automatically generate a learning curriculum (Figure 1a). That is, we consider Alice and Bob as two versions of the same underlying agent, and each agent's objective is the negative of the others', so $R^A = -R^B$. However, this approach is only applicable for symmetric zero-sum games, and does not lend itself to learning curricula for goal-conditioned agents in other setups.

Asymmetric Self Play (ASP). ASP [40] extends self play to non-zero sum games by proposing an asymmetric parametrization of two competing agents. During each episode, Alice's objective is to propose and demonstrate a task for Bob to complete (Figure 1b). In the goal-conditioned paradigm, we take Alice's final state s_T^A as the proposed goal g. Alice and Bob now have different objectives: Bob's return, $R^B = \sum_t \gamma^t r(s_t^B|g)$, is simply based on reaching the proposed goal, while Alice's reward is based on Bob's inability to reach the goal, $R^A = \mathbb{1}(d(s_T^B, g) \geq \epsilon)$. While this approach lends itself well to automatic curriculum generation in more complex tasks such as robotic manipulation [28], it has poor sample efficiency. Since we rely on a learning agent, Alice, to propose and solve goals, the generated goals are limited by Alice's own ability to explore and learn challenging yet feasible goals. This can be particularly problematic towards the beginning of training, where we can waste many episodes proposing simple and redundant goals for Bob.

3 Automatic Curriculum Generation via CuSP

Our curriculum generation method consists of four main players: two goal-conditioned learning peers, Alice π_A and Bob π_B, and two goal generators, G_A and G_B. $\pi_A(a|s, g)$ and $\pi_B(a|s, g)$ are both optimized to maximize the discounted sum of goal-conditioned rewards $R(g) = \sum_t \gamma^t r^t(g)$, across proposed goals $g \in \mathcal{G}_{id}$. Aside from any initial randomness in initializing π_A and π_B, the two policies are parameterized with the same architecture. $G_A(g|s_0, z)$ and $G_B(g|s_0, z)$ are parametrized as policies that output a goal $g \in \mathcal{G}_{id}$ given an initial observation of the environment s_0 and a latent noise variable $z \sim \mathcal{N}(0, 1)$. The inclusion of such a noise variable draws from work in generative models (e.g. GANs, VAEs) where such a z is the input to the generator, and is also used in prior work in goal generative models [8]. The goal generators are optimized to maximize the regret of their corresponding agent: $\mathfrak{R}^{G_A} = R^A(g) - R^B(g)$ and $\mathfrak{R}^{G_B} = -\mathfrak{R}^{G_A} = R^B(g) - R^A(g)$.

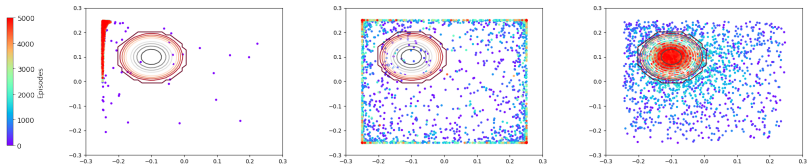
We refer to our overall curriculum generation approach as *Curriculum Self Play* (CuSP). We present an overview of CuSP in Algorithm 1 and illustrate it in Figure 1c. The algorithm proceeds in rounds. In every round, both goal generators propose a goal, g_A from G_A and g_B from G_B. To keep the symmetrization fair and hopefully scaffold learning initially, we first rollout the 'easier' goal for each respective agent (ie. g_A for Alice, g_B for Bob) and compute the corresponding environment returns and update the learners. Next, we rollout the 'harder' goal for each respective agent (ie. g_B for Alice, g_A for Bob) and update the learners again. At the end of the round, we update the goal generators with the respective regrets.

Symmetrization. The regret objective $R^A - R^B$ helps G_A propose goals that are challenging for Bob while still being feasible for Alice. However, this can be detrimental when Alice and Bob's

Algorithm 1: Curriculum Self Play (CuSP)

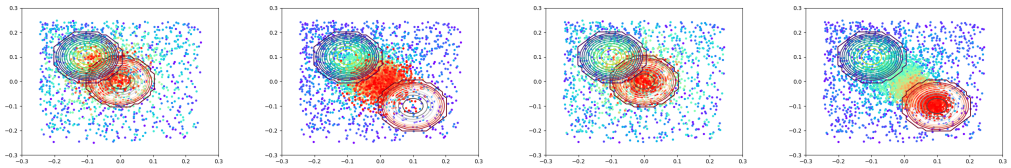
Initialize goal-conditioned learners π_A, π_B, goal generators G_A, G_B;
while *not converged* **do**
  `// generate goals`
  $g_a \sim G_A, g_b \sim G_B$;
  `// evaluate easier goals first`
  $R^A_{Easy} \leftarrow rollout(\pi_A, g_A)$; $R^B_{Easy} \leftarrow rollout(\pi_B, g_B)$;
  `// update policies`
  $\pi_A \leftarrow train_learner(\pi_A, R^A_{Easy})$; $\pi_B \leftarrow train_learner(\pi_B, R^B_{Easy})$;
  `// evaluate harder goals second`
  $R^A_{Hard} \leftarrow rollout(\pi_A, g_B)$; $R^B_{Hard} \leftarrow rollout(\pi_B, g_A)$;
  `// update policies`
  $\pi_A \leftarrow train_learner(\pi_A, R^A_{Hard})$; $\pi_B \leftarrow train_learner(\pi_B, R^B_{Hard})$;
  `// compute regrets and update generators`
  $G_A \leftarrow train_generator(G_A, \{(R^A_{Easy} - R^B_{Hard}), g_A\}, \{(R^A_{Hard} - R^B_{Easy}), g_B\})$;
  $G_B \leftarrow train_generator(G_B, \{(R^B_{Easy} - R^A_{Hard}), g_B\}, \{(R^B_{Hard} - R^A_{Easy}), g_A\})$;
end

performance diverge too quickly. To address this, we propose symmetrizing the multi-agent system by introducing a corresponding goal generator G_B whose regret objective is $R^B - R^A$, essentially pitting it in a zero-sum game against G_A. Our insight is that having these two goal generators produces two types of goals for each agent: a) a feasible yet not too simple goal from the 'friendly' goal generator (G_A for Alice, G_B for Bob), which can help scaffold learning quickly, and b) a challenging yet not too difficult goal from the 'unfriendly' goal generator (G_A for Bob, G_B for Alice), which can help push the boundaries of the agent's abilities. Both of these properties help encourage the generation of goals that are **feasible** yet progressively challenging.



(a) Adam (b) PPO (c) SAC

Figure 2: Performance of different optimization methods on a toy landscape that is primarily flat except for a single gaussian centered at (-0.1, 0.1) shown by the contours. Red shows most recently proposed points. SAC is able to find the peak while the other methods get stuck in the flat regions.



(a) SAC *without* regret updates at 2500 and 5000 steps. (b) SAC *with* regret updates at 2500 and 5000 steps.

Figure 3: SAC optimizing a non-stationary toy landscape that is primarily flat, but has a gaussian peak starting at (-0.1, 0.1) and moving diagonally to (0.1, -0.1). Red shows most recently proposed points. Without the regret updates, the optimizer lags and cannot find the optimum.

Goal Generator Design. We motivate our goal generator design with our proposed desiderata for curriculum generation: progressive exploration balanced with anti-catastrophic exploitation. That is, we want to propose goals that are progressively more challenging, but also prioritize revisiting goals

that may have been forgotten by the learners. Although the goal generator is not an RL agent in the traditional sense – it only 'acts' in a single timestep by proposing a goal and its 'observation' only changes based on a random noise variable – optimizing such an explore-exploit tradeoff effectively naturally inspires the use of an RL objective. RL objectives have been used for one-step optimization in other contexts as well (e.g., one-step Q-learning [45]).

Thus, to train the goal generators we use an RL learning objective and parameterization inspired by a single-time step variant of Soft-Actor Critic (SAC) [13]. SAC is an off policy algorithm that optimizes for a trade-off between expected reward and policy entropy $H(\pi)$ with a coefficient α. In our setup, our goal generator only acts once to propose a goal per episode, so each 'trajectory' is only a single time step consisting of the initial state s (for which we concatenate the initial environment observation s_0 with a latent noise variable z), the proposed goal action g, and we get the corresponding "reward" for the goal generator policy by computing the regret between π_A and π_B on g. Thus the objective for each goal generator policy G is given by

$$\max_G \mathbb{E}_{g\sim G}\left[\underbrace{\mathfrak{R}^G(s,g)}_{\text{exploitation}} + \alpha \underbrace{H(G)}_{\text{exploration}}\right]$$

where the regret term addresses anti-catastrophic exploitation by prioritizing goals where regret has increased, and the entropy term addresses progressive goal space exploration.

We equip the SAC goal generators with a replay buffer to store all past goals and their regrets. This helps address **forgetting** and improve the sample efficiency of goal generation compared to prior methods which optimize the generator as another agent using on-policy algorithms such as PPO [34]. We further take advantage of SAC's entropy regularization to progressively increase the **diversity** of the goals, with consideration of Alice and Bob's capabilities using the stored regrets. To empirically motivate the use of SAC, we compare SAC against PPO and Adam [16] in optimizing a synthetic landscape. One challenge of the regret objective is that at the beginning of training, the landscape will mostly be flat when π_A and π_B are similarly unskilled. In Fig. 2, we find that only SAC is proficient at optimizing a primarily flat landscape.

Dynamic Regret Updates for Non-stationary Agents. However, using an off policy RL algorithm for optimizing such a goal generator requires further modifications. For more typical agents, the received rewards are a stationary property of the environment. In our case this is no longer true – Alice and Bob are continuously learning and part of the multi-agent environment, so the regrets for the same goal will also change over time. To address this issue, we would like to update the non-stationary regret values in the goal generator's replay buffer. However, it is expensive for Alice and Bob to replay all the goals in the environment. As a sample efficient trick, we leverage the observation that the critics of Alice and Bob can provide a current estimate of their performance. Hence, after every round, we update the regrets of previous goals in the replay buffer using the difference between the corresponding value estimates obtained via the critic networks of Alice and Bob. This allows us to keep an updated estimate of goal **feasibility** based on Alice and Bob's current skills.

To empirically validate that updating the replay buffer regrets helps in a non-stationary landscape, we carry out investigations of a SAC goal generator both with and without regret updates in Fig. 3. As hypothesized, updating the replay buffer allows the generator to find the moving optimum.

The combination of the replay buffer and dynamic regret updates also strengthens our method by automatically prioritizing goals that have been forgotten. With the non-stationary regret updates, the goal generators always have an updated estimate of each learner's abilities. Thus if Bob performs worse on a past goal, the regret increases for G^A and it is more likely to repropose it, addressing catastrophic **forgetting** for Bob. Likewise, symmetrization helps to address catastrophic **forgetting** for Alice in the same manner. Without the corresponding G_B, if Alice performs worse at a goal, the regret $R^A - R^B$ decreases and G_A is less likely to repropose it unless Bob's performance decreases even further. This can potentially lead to neglecting to propose these challenging goals for Bob.

Summary. Inspired by Racaniere et al. [32], we specify desired qualities of a goal generator:

- **Progressive Feasibility** – the goals should be incrementally more difficult, accounting for the agent's current skill level and the fact that the agent is a non-stationary learner.

Table 1: Contribution of each component of our method to the desired qualities.

	Progressive Feasibility	**Progressive Diversity**	**Anti-Forgetting**
SAC Entropy Regularization		✓	
Regret Replay Buffer			✓
Dynamic Regret Updates	✓		✓
Symmetrization	✓		✓

- **Progressive Diversity** – the goals should encourage increasing exploration of goals.
- **Anti-Forgetting** – if previous goals are forgotten, they should be reproposed.

We summarize the contributions of each component of our method to desired qualities of feasibility, diversity, and anti-forgetting in Table 1. We visualize generated goal distributions in Appendix A.

4 Experiments

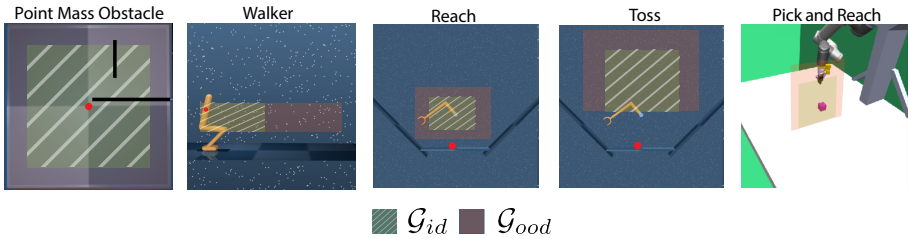


Figure 4: Illustration of the five environments, $\mathcal{G}_{id}$ showing training goal space and $\mathcal{G}_{ood}$ showing test goal space. The red points indicate what is used to evaluate whether a goal was reached in each environment – the point mass coordinates for Point Mass Obstacle, the torso center location for Walker, the ball location for Reach/Toss, and the block location for Pick and Reach.

Environments. We test our method across a suite of continuous control tasks adapted from the Deepmind Control Suite [41] and OpenAI Gym [29], using the MuJoCo simulator [42]. We analyze our method across a variety of task types: *navigation*, *locomotion*, and *manipulation*. Point Mass Obstacle is a navigation task with some walls. Walker is a locomotion task where success is achieved when the torso reaches the goal. Reach and Toss are manipulation tasks where success is achieved when moving a ball that starts in the gripper to the goal. The latter environment provides an interesting challenge as gravity plays a large role – the robot must learn to aim precisely. Pick and Reach [26] is a manipulation task where success is measured by picking up a block and moving it to a target position. Environment specifics can be found in Appendix B.2.

Training Setup. For CuSP, we train the goal generators using the SAC implementation from Yarats and Kostrikov [47] with their default architecture and hyperparameters. We compare against a uniform random generator baseline and two other baselines that use minimax formulations for curriculum generation: the GoalGAN generator-discriminator approach [8] and ASP+BC [28]. For all methods we train the learner policies with the same architecture and SAC to isolate differences to the goal generation method. Code and further training details can be found in the Supplementary Materials.

We aim to answer the following questions with our experiments:

- Does the CuSP curricula induce agents that can generalize to novel goals from $\mathcal{G}_{ood}$?
- Does the CuSP curricula induce interesting emergent skills for solving harder goals?
- Does the CuSP curricula handle cases where portions of the goal space are poorly specified (e.g. with impossible dimensions)?

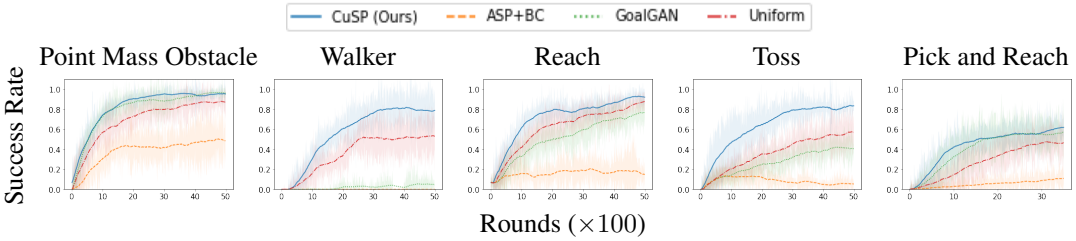


Figure 5: Success rates across environments on random goals sampled from $\mathcal{G}_{ood}$. Results averaged across 3 seeds for each method. Each round corresponds to goal generation, then rollouts, then the respective goal generator updates. Generally we find our method matches or outperforms all baselines in out-of-distribution generalization, especially when tackling the harder locomotion task.

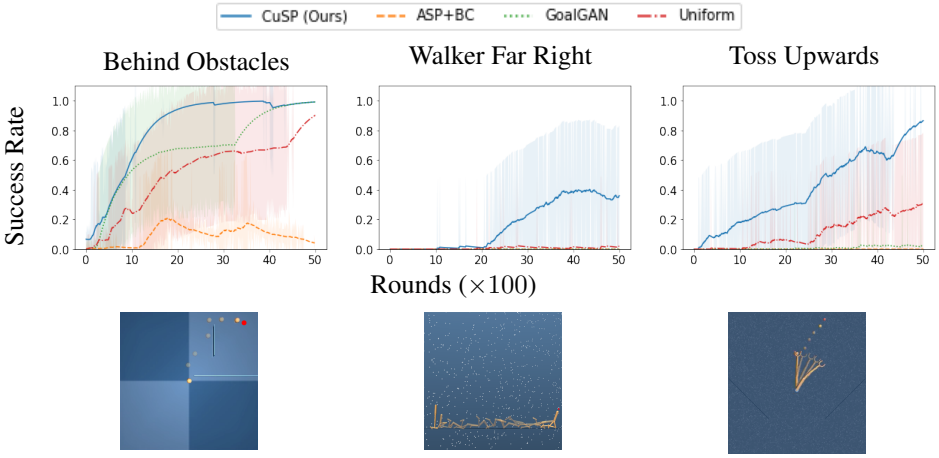


Figure 6: Success rates for three more challenging goals in Point Mass Obstacle, Walker, and Manipulator, and corresponding sample trajectories. In Point Mass we learn to navigate around walls faster than other methods, in Walker we learn to run quickly (albeit by using its torso) whereas other methods fail to succeed at all, and in Toss we learn to aim upwards accurately.

4.1 Test-Time Out-Of-Distribution Generalization

To assess whether our curriculum improves generalizability, we evaluate each method on its ability to reach randomly sampled out-of-distribution goals $g \sim \mathcal{G}_{ood}$. In Fig. 5 we see that across environments, generally CuSP matches or outperforms all the baselines for task success. ASP+BC's sample inefficiency due to Alice's limitations leads to poor performance, while GoalGAN does well in environments where the goal space is more symmetrical (e.g. with navigation or the reaching tasks, where most points in the goal space are similarly difficult). That said, we note GoalGAN struggles more in environments where portions of the goal space are more challenging (e.g. walking to the right or aiming and tossing upwards). We hypothesize this is due to the challenges of accurately learning the feasibility of different goals in an asymmetric goal space. Lastly, we find that the uniform random baseline is particularly well suited for this OOD random evaluation. The baseline is trained on good coverage of the in-distribution goal space, which is beneficial for generalization to random goals in the navigation and manipulation tasks, but is less helpful for the more asymmetric goal spaces like Walker and Toss. For ablations on the components of our method, see Appendix B.5.1 and for ablations on ASP, see Appendix B.5.2.

4.2 Task-Specific Skills

Next, we examine if CuSP can induce emergent skills in learners for tackling harder, environment specific goals. Below, we enlist 3 representative examples in the more asymmetric environments

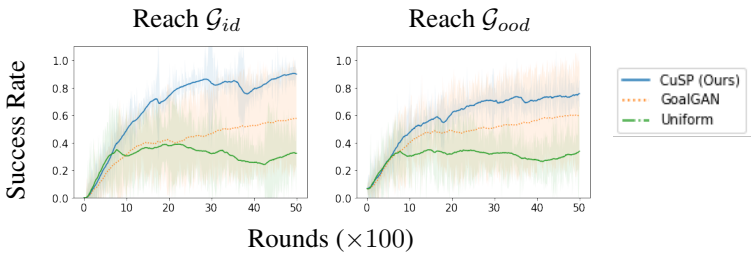


Figure 7: Success rates for Reach with a third impossible dimension added to the specified goal space. We test on both in distribution and out of distribution randomly sampled goals in the 3D goal space, where the third dimension is not feasible in the environment. Generally we find CuSP is least affected while the baselines generally performs worse with the higher dimensional space.

where certain goals are clearly more challenging to solve. As seen in Fig. 6, our method is able to improve upon success rates on these goals.

- Point Mass Obstacle – *Behind Obstacles*: requires learning to move around the walls to the edge of the training space. Here, we learn to succeed much faster than the other methods.
- Walker – *Far Right*: requires learning to move forward quickly to reach a goal far out of the training space. Here, we learn to succeed at the task where all other baselines fail, suggesting that our method produces a useful curriculum for learning challenging locomotion tasks while the others cannot, or require many more samples to do so.
- Toss – *Upwards*: requires learning to toss the ball accurately upwards outside of the training space. Here, we learn to succeed much faster than the other methods.

4.3 Misspecified Goal Dimensions

Lastly, we investigate if the goal generator methods (CuSP, GoalGAN, and Uniform) can handle the case where parts of the specified goal space is misspecified. To do this, we append an extra dimension to the goal space, bounded between $[-1, 1]$, and evaluate performance on $\mathcal{G}_{id}$ and $\mathcal{G}_{ood}$ with randomly sampling the third dimension during evaluation. We use Reach as a case study, and as seen in Fig. 7, agents trained using CuSP's goal curricula are less affected by the increased goal space while the baselines plateau at a lower success rate. In particular, the Uniform baseline is most adversely affected by the higher dimensionality. This suggests that while random sampling provides good coverage and is amenable to more simple goal spaces, as seen above, the induced curricula degrades in quality with increasingly complex goal spaces. Note that we exclude ASP from these experiments as it does not use a pre-specified goal space and only sets achieved states as goals.

5 Related Work

We summarize four main thrusts of work within automatic curriculum generation.

Modeling Goal Distributions. Many previous works have tackled the problem of generating "appropriately" difficult goals for an autocurriculum. In discrete environments, Campero et al. [5] propose an adversarial goal-generator that is rewarded when the learning agent achieves the goal with a thresholded level of effort. However, they rely on a heuristic time limit hyperparameter to determine 'suitable effort' (i.e. goal feasibility) and hyperparameters for the teacher reward, whereas our approach directly uses regrets from multiagent interactions to automatically gauge feasibility and provide a reward signal to the teacher. In continuous domains, Zhang et al. [48] samples goals that maximize the uncertainty of the learner's Q-function, and Nair et al. [24] learn a latent representation to sample 'imagined' goals from. GoalGAN [8] trains a GAN to generate goals of intermediate difficulty for the agent, using a discriminator to predict whether the generated goals would be appropriate by thresholding success rates. These methods use a heuristic measure of challenging goals, while we leverage the multi-agent setting to automatically find interesting goals. Similarly to GoalGAN, the Setter-Solver paradigm [32] trains a goal setter whose objectives are a balance of goal validity, feasibility, and coverage, as well as a discriminative judge for predicting goal feasibility. Our

approach differs in that we do not need to explicitly weigh the contributions of each loss, and we do not need an accurate discriminator for estimating feasibility.

Multi-Agent Curriculum Learning. The idea that interactions between multiple agents can lead to emergent curricula has been proposed in prior works [19, 2] and can be connected with how humans learn through social interactions. One parametrization of such interactions is through designating another agent(s) as a teacher who aims to select an appropriate curriculum for the student. For task-based curricula, Matiisen et al. [23] train a teacher policy for selecting sub-tasks for the student, and Graves et al. [11], Portelas et al. [30] treat task selection as a bandit problem for the teacher.

Rather than having an explicit teacher, Asymmetric Self-Play (ASP) [40] automatically generates a curriculum for exploration by training a similarly parametrized agent, Alice, to demonstrate the easiest possible task that the learning agent, Bob, can not yet complete. OpenAI et al. [28] extend ASP to robotic manipulation tasks by incorporating an imitation learning loss for Bob to imitate Alice on challenging goals. As Alice is no longer setting the goals, we do not make use of behaviour cloning in CuSP. While they demonstrate zero-shot generalization to a diverse set of target tasks, relying on Alice to set the goals can also be incredibly sample inefficient, as we saw in our evaluations. Sodhani and Pahuja [38] propose a memory-augmented version of ASP by providing the learners an external memory to speed up exploration, which is analogous to our method training with off-policy learners. Sukhbaatar et al. [39] extends ASP to a hierarchical learning framework by learning subgoal representations. Competitive Experience Replay [21] generates an exploratory competitive curriculum through penalizing Alice if visiting states Bob has visited, but rewards Bob for visiting states found by Alice.

Environment Design. Another method for inducing curricula is through generating a sequence of environments to learn in. Procedural level generation is a common approach for developing games, and can lead to increased generalizability [15]. POET [44] uses a population of adversaries to generate increasingly challenging environments for a population of learning agents. PAIRED [6] proposes optimizing an adversarial environment designer through minimax regret between two learning agents, inspiring the objective we use for our goal generators. However, this approach does not account for catastrophic forgetting and can suffer from sample inefficiencies from using an on-policy method and in cases where the regret landscape is flat. That said, environment design approaches can be used in a complementary way with our approach, extending it beyond goal generation.

Skill Discovery. A closely related method for curriculum generation is through unsupervised skill discovery, where an agent explores an environment to automatically discover skills that can be composed to achieve a variety of tasks. Prior methods have used information theoretic objectives to encourage skill acquisition that covers a diverse set of behaviours [7] or generate a multi-task distribution [12]. For robotics tasks, [14] propose a method for learning a latent skill space that can be composed and is transferable between tasks, DADS [36] proposes a combination of model-based and model-free RL that optimizes for discovering predictable skills that can be composed by a planner, and Play-LMP [22] proposes using self-supervision across play data for skills discovery. Another method for curriculum generation is through using hindsight [1, 20]. These works are complementary to ours as our method aims to induce a curriculum through sparse goal generation, which can be augmented with unsupervised skill discovery.

6 Conclusion

We propose a new method, CuSP, for automatic curriculum generation based on symmetric extension of self play approaches for non zero-sum games. Our proposed method aims to address weaknesses of prior approaches by being designed to specifically consider a balance of progressive exploration and anti-catastrophic exploitation. To this end, we make use of two off-policy students and two off-policy regret-maximizing goal generators, each of which is more 'friendly' to one of the students. As a result, we generate goals that are a) challenging yet feasible, b) diverse, and c) are anti-forgetting. We qualitatively highlight these desired qualities in goal generation compared to prior methods, demonstrate quantitative gains at generalizing to out-of-distribution unseen goals, and find some emergent task-specific skills.

Limitations and Future Work. While our method shows promise for addressing key components of generating useful curriculum, there is still significant room for improvement. One limitation of our current approach is that it depends on specifying a goal space for the generator to act in, which was

not necessary in the original ASP method. That said, specifying a goal space also allows us to imbue the system with human priors for what goals should or should not be valid, whereas ASP may need post-hoc goal filtering if Alice ends up in an undesirable state.

To further tackle automatic curriculum generation, future work should explore scenarios where goal spaces are not fully pre-specified, as well as extending to high dimensional goal spaces (e.g. images). Furthermore, another interesting avenue would be extrapolating CuSP to a larger number of student and teacher agents, which can potentially provide benefits through better goal space coverage and giving a more accurate estimate of feasibility.

References

[1] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba. Hindsight experience replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf`.

[2] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch. Emergent tool use from multi-agent autocurricula. In *International Conference on Learning Representations*, 2019.

[3] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch. Emergent complexity via multi-agent competition. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Sy0GnUxCb`.

[4] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48, 2009.

[5] A. Campero, R. Raileanu, H. Kuttler, J. B. Tenenbaum, T. Rocktäschel, and E. Grefenstette. Learning with {amig}o: Adversarially motivated intrinsic goals. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=ETBc_MIMgoX`.

[6] M. Dennis, N. Jaques, E. Vinitsky, A. Bayen, S. Russell, A. Critch, and S. Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13049–13061. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper/2020/file/985e9a46e10005356bbaf194249f6856-Paper.pdf`.

[7] B. Eysenbach, A. Gupta, J. Ibarz, and S. Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2018.

[8] C. Florensa, D. Held, X. Geng, and P. Abbeel. Automatic goal generation for reinforcement learning agents. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1515–1528. PMLR, 10–15 Jul 2018.

[9] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[10] I. L. Glicksberg. A further generalization of the kakutani fixed point theorem, with application to nash equilibrium points. *Proceedings of the American Mathematical Society*, 3(1):170–174, 1952.

[11] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pages 1311–1320. PMLR, 2017.

[12] A. Gupta, B. Eysenbach, C. Finn, and S. Levine. Unsupervised meta-learning for reinforcement learning, 2020.

[13] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870. PMLR, 10–15 Jul 2018.

[14] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rk07ZXZRb.

[15] N. Justesen, R. R. Torrado, P. Bontrager, A. Khalifa, J. Togelius, and S. Risi. Illuminating generalization in deep reinforcement learning through procedural level generation, 2018.

[16] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In Y. Bengio and Y. LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[17] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[18] I. Kostrikov. Pytorch implementations of reinforcement learning algorithms. https://github.com/ikostrikov/pytorch-a2c-ppo-acktr-gail, 2018.

[19] J. Z. Leibo, E. Hughes, M. Lanctot, and T. Graepel. Autocurricula and the emergence of innovation from social interaction: A manifesto for multi-agent intelligence research, 2019.

[20] A. Li, L. Pinto, and P. Abbeel. Generalized hindsight for reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

[21] H. Liu, A. Trott, R. Socher, and C. Xiong. Competitive experience replay. In *International Conference on Learning Representations*, 2018.

[22] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In L. P. Kaelbling, D. Kragic, and K. Sugiura, editors, *Proceedings of the Conference on Robot Learning*, volume 100 of *Proceedings of Machine Learning Research*, pages 1113–1132. PMLR, 30 Oct–01 Nov 2020. URL http://proceedings.mlr.press/v100/lynch20a.html.

[23] T. Matiisen, A. Oliver, T. Cohen, and J. Schulman. Teacher–student curriculum learning. *IEEE transactions on neural networks and learning systems*, 31(9):3732–3740, 2019.

[24] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. *Advances in Neural Information Processing Systems*, 31:9191–9200, 2018.

[25] J. Nash. Non-cooperative games. *Annals of mathematics*, pages 286–295, 1951.

[26] OpenAI. Robogym. https://github.com/openai/robogym, 2020.

[27] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, J. Schneider, N. Tezak, J. Tworek, P. Welinder, L. Weng, Q. Yuan, W. Zaremba, and L. Zhang. Solving rubik's cube with a robot hand, 2019.

[28] O. OpenAI, M. Plappert, R. Sampedro, T. Xu, I. Akkaya, V. Kosaraju, P. Welinder, R. D'Sa, A. Petron, H. P. d. O. Pinto, A. Paino, H. Noh, L. Weng, Q. Yuan, C. Chu, and W. Zaremba. Asymmetric self-play for automatic goal discovery in robotic manipulation, 2021.

[29] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018.

[30] R. Portelas, C. Colas, K. Hofmann, and P.-Y. Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments, 2019.

[31] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer. Automatic curriculum learning for deep rl: A short survey. *arXiv preprint arXiv:2003.04664*, 2020.

[32] S. Racaniere, A. Lampinen, A. Santoro, D. Reichert, V. Firoiu, and T. Lillicrap. Automated curriculum generation through setter-solver interactions. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=H1e0Wp4KvH.

[33] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, and et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, Dec 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-03051-4. URL http://dx.doi.org/10.1038/s41586-020-03051-4.

[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[35] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[36] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2019.

[37] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[38] S. Sodhani and V. Pahuja. Memory augmented self-play, 2018.

[39] S. Sukhbaatar, E. Denton, A. Szlam, and R. Fergus. Learning goal embeddings via self-play for hierarchical reinforcement learning, 2018.

[40] S. Sukhbaatar, Z. Lin, I. Kostrikov, G. Synnaeve, A. Szlam, and R. Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations*, 2018.

[41] Y. Tassa, S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, and N. Heess. dm_control: Software and tasks for continuous control, 2020.

[42] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

[43] A. Tychonoff. Über die topologische erweiterung von räumen. *Mathematische Annalen*, 102(1): 544–561, 1930.

[44] R. Wang, J. Lehman, J. Clune, and K. O. Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions, 2019.

[45] C. J. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[46] S. Willard. *General topology*. Courier Corporation, 2012.

[47] D. Yarats and I. Kostrikov. Soft actor-critic (sac) implementation in pytorch. `https://github.com/denisyarats/pytorch_sac`, 2020.

[48] Y. Zhang, P. Abbeel, and L. Pinto. Automatic curriculum learning through value disagreement. *Advances in Neural Information Processing Systems*, 33, 2020.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

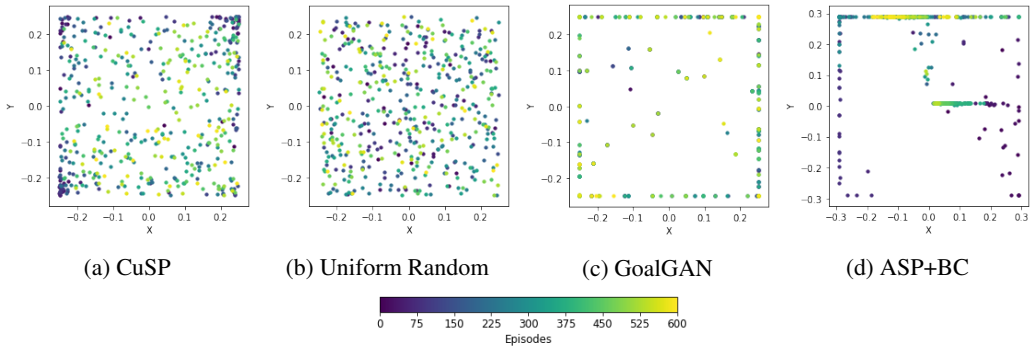
1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6 for a discussion on limitations.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] No direct ties to applications.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplemental material for codebase.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 4 for architecture and training details.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 4 evaluation plots with error bars, and we run all our experiments across three seeds.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Supplementary Materials for compute details.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] The SAC implementation we build off of is Yarats and Kostrikov [47], the PPO implementation we build off if os [18] and we additionally use the simulated environments of Tassa et al. [41] and Plappert et al. [29] , cited in Section 4.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include our codebase in the supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Desired Goal Generation Qualities

We begin by qualitatively examining the key properties of the curriculum of goals generated in Point Mass Obstacle. We see that qualitatively our method (Fig. 8a) proposes goals with more progressive exploration than either ASP (Fig. 8d) (which seems to get stuck on the walls, limited by Alice's ability to explore) or GoalGAN (Fig. 8c) (which seems to get stuck more along the borders of the space, only prioritizing difficult goals). The uniform random generator (Fig. 8b) has the opposite result, having great coverage but without any consideration for feasibility or difficulty. Our method manages to balance the exploration with progressive feasibility, where we see CuSP proposing goals more towards the corners at first, then progressively moving towards the center and covering the space more thoroughly, with slightly more density in the top right where the maze is. We hypothesize there is more concentration in the corners initially when the goal generators are untrained and proposing goals towards the edges of the goal constraints.



(a) CuSP (b) Uniform Random (c) GoalGAN (d) ASP+BC

Figure 8: Progressive goal generation plots for Point Mass Obstacle after 500 rounds across all four methods. We see more progressive diversity from CuSP, whereas GoalGAN mostly stays on the outer border and ASP+BC gets stuck on the environment walls, limited by Alice's abilities.

B Experiments

Compute. To train each method in each environment, we use 1 NVIDIA V100 GPU per seed in an internal cluster. This amounts to an approximate total of 400 GPUs for the final evaluations and hyperparameter sweeps. We additionally did extensive prototyping on the Point Mass environment before scaling to the more complex environments, but do not have a reasonable estimate for how much compute was expended in that phase of the project.

B.1 Toy Experiments

For the toy evaluations in Figures 2 and 3, we evaluate on the landscape

$$\text{regret}(x, y) = \begin{cases} -((x+.1)^2 + (y-.1)^2) & \text{if} (x+.1)^2 + (y-.1)^2 < .01 \\ -.01 & \text{otherwise} \end{cases}$$

For Adam, we use the default configuration from PyTorch (lr=0.001, βs = (0.9, 0.999), ϵ=1e-8, no weight decay). For PPO, we use the default configuration from [18]. We found our trends to hold robustly for a fairly broad choice of lr's and both with/without GAE [35]. In the toy experiments, we carry out 5000 steps of goal proposal with one gradient update for every method at each step. For the non-stationary experiment in Figure 3, we diagonally perturb the center of the regret landscape from top left quadrant (initialized at (-0.1, 0.1)) to the bottom right quadrant at a constant rate of 2e-4 every round. Hence, at the end of 2500 rounds, the center of the landscape is at the origin and at the end of 5000 rounds, the center of the landscape is at (0.1, -0.1).

B.2 Environment Specifics

Table 2 describes the reward configurations we used in our environments. For the manipulation environments, we also incorporated HER [1]. ϵ is the tolerance for success (i.e. a goal g is successfully achieved at state s if the normed distance $L_2(s-g) < \epsilon$). We terminate the episode early if successful.

Env	Reward	Use HER [1]	ϵ
Point Mass Maze	-L_2		.05
Walker	-L_2		.1
Reach	-L_2	✓	.1
Toss	-L_2	✓	.1
Pick and Reach	0 if at goal, -1 otherwise	✓	.1

Table 2: Environment Reward Configurations

We modify the default Point Mass environment by adding in walls to the top right corner and initialize the agent in between the walls 10% of the time to help with exploration. We modify the manipulator environment by always initializing with the ball in the gripper and with the gripper in an upright position. We use an episode length of 1000 steps for Point Mass Obstacle and Walker and an episode length of 100 steps for Reach, Toss, and Pick.

Specific definitions of in-distribution and out-of-distribution goal spaces for each environment are in Table 3

Env	$\mathcal{G}_{id}$	$\mathcal{G}_{ood}$
Point Mass Maze	$[-.25, .25] \times [-.25, .25]$	$[-.3, .3] \times [-.3, .3]$
Walker	$[-1, 10] \times [-.2, .2]$	$[10, 20] \times [-.2, .2]$
Reach	$[-.25, .25] \times [.2, .6]$	$[-.4, .4] \times [.1, .8]$
Toss	$[-.5, .5] \times [.5, 1.3]$	$[-.6, .6] \times [.5, 1.4]$
Pick and Reach	$[.5, .9] \times [.5, .7]$	$[.4, 1] \times [.5, .8]$

Table 3: Goal spaces for each environment.

B.3 Hyperparameters

We train the learners and goal generators using the default SAC configuration and implementation from [47], as listed in Table 4.

Parameter	Value
γ	.99
Initial α	.1
α LR	1e-4
Actor LR	1e-4
Critic LR	1e-4
Batch size	1024
Critic hidden dim	1024
Critic hidden depth	2
Actor hidden dim	1024
Actor hidden depth	2

Table 4: SAC Hyperparameters

To scale the output of the goal generators to the correct action space, we take the Tanh scaled output and rescale each dimension to the goal space. We use Appendix C of [13] to modify the log loss for SAC accordingly. To train the goal generator, we update after each round (i.e. per goal proposed) with 100 gradient updates. As the learners are updated once per time-step in the environment (100-1000 updates depending on the environment), we increased the update frequency for the goal-generator to keep pace with the learner updates since the goal generator operates on a single time step per episode.

B.4 Baselines

We reimplement each baseline by comparing against any available public code and matching each paper's recommended training hyperparameters as closely as possible within our compute limitations.

Off vs. On Policy Learners In ASP+BC[28], Alice and Bob are trained using PPO. We also implemented this as a baseline but found that the agents were unable to reach high success at the same rate as our SAC-trained agents. For a comparison that's more favorable to the baselines, we report the success rates from training with the same agent configuration (ie. Bob trained with SAC) in the main paper to isolate learning differences to the goal generation method.

B.5 Ablations

B.5.1 CuSP Ablations

SAC Motivation Without symmetrization, the regret-based goal generation objective is the same as the proposed objective in PAIRED [6]. However, our contribution extends beyond the symmetrized objective as our proposed method reframes the goal generation process into a entropy-regularized agent with a memory buffer – which motivated designing our method around SAC. PAIRED uses PPO to optimize their environment generators, which we found was not sample efficient enough for our tasks (i.e. success rates were too low), as our toy experiment in Figure 2 suggested.

Here we highlight a case study with the Toss task into why the entropy regularization and replay buffer components are important. Building up from just the regret objective as in PAIRED, we have in Figure 9:

1. $\beta = 1, \alpha = 0$, no replay buffer – Regret objective only for a single goal generator, as in PAIRED.

2. $\beta = 1, \alpha = 0$, no replay buffer, symmetrized – Regret objective with two symmetrized goal generators. Symmetrization seems to improve performance slightly, but is not sufficient alone.

3. $\beta = 1, \alpha = 0$ – Regret objective with a replay buffer for the goal generator. Also improves performance slightly, although not as much as symmetrization.

4. $\beta = 1$ – Regret objective with a replay buffer and entropy regularization. This is particularly helpful for the random OOD evaluation as the greater goal diversity improves multi-goal performance on the multi-goal evaluation task, as expected.

5. $\beta = 1$ symmetrized – Regret objective with a replay buffer and entropy regularization, and symmetrized goal generators.

6. $T = 300, \beta = .1$, symmetrized (Full CuSP) – Regret objective with a dynamically updated replay buffer and entropy regularization, and symmetrized goal generators.

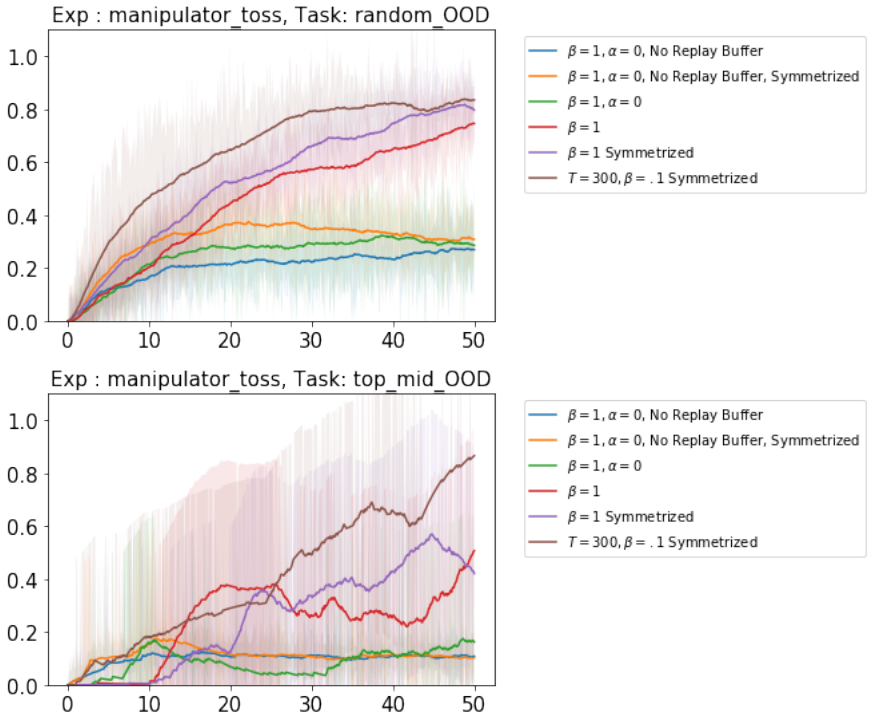


Figure 9: Full ablations of all components of CuSP from Table 1. Success rates are averaged across 3 seeds.

Looking at a sample of the generated goals in Figure 10, we find a stark qualitative improvement in goal diversity between the regret-only PAIRED condition and CuSP. In particular, in the baseline the generated goals converge only on the corners.

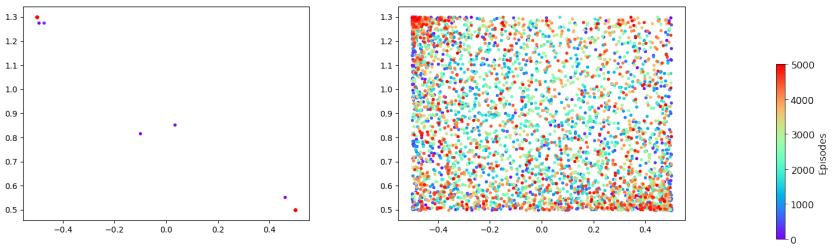


Figure 10: Goal plots after 2000 rounds for Toss. The left plot shows the goals generated in the $\beta = 1, \alpha = 0$, No Replay Buffer condition and the right plot shows the goals generated in the $T = 300, \beta = .1$, Symmetrized condition from Figure 9. Red indicates more recent goals.

Regret update and Symmetrization Ablations As we use the learners' critics for estimating the current regret for a particular goal, we need to be careful about how we use the critic data since it can be inaccurate, especially towards the beginning of training. We introduce two hyperparameters: one for specifying at which episode we begin regret updates T, and a weighting parameter β for how much to update the regrets by. We use the update rule

$$\text{regret} = \beta \cdot \text{old regret} + (1 - \beta) \cdot \text{new regret}$$

We sweep across $T = 50, 100, 200$ and $\beta = .1, .5, .9$ to select the best performing hyperparameters based on average performance on the training goal set, $\mathcal{G}_{id}$. In Figure 11, we ablate over the different components of our method: CuSP+SAC with no entropy regularization ($\beta = 1, \alpha = 0$), CuSP+SAC only ($\beta = 1$), CuSP+SAC+Symmetrization ($\beta = 1$), and CuSP+SAC+Regret

Updates+Symmetrization to investigate the effects of each component. In most of the tasks, aside from Reach on random out of distribution goals, symmetrization improves performance. In particular, we see larger gains on tasks where $\beta = 1$ performance is lower to begin with, such as Walker. Further gains from stale regret updates vary from being very small on tasks where $\beta = 1$ performance is already high (eg. for Point Mass Obstacles or Toss on random OOD goals), to larger improvements on the harder skill tasks (eg. Behind Obstacles or Walker Far Right goals). Interestingly, we also note that while entropy regularization is crucial for most of the environments, setting $\alpha = 0$ seems to be helpful for Walker.

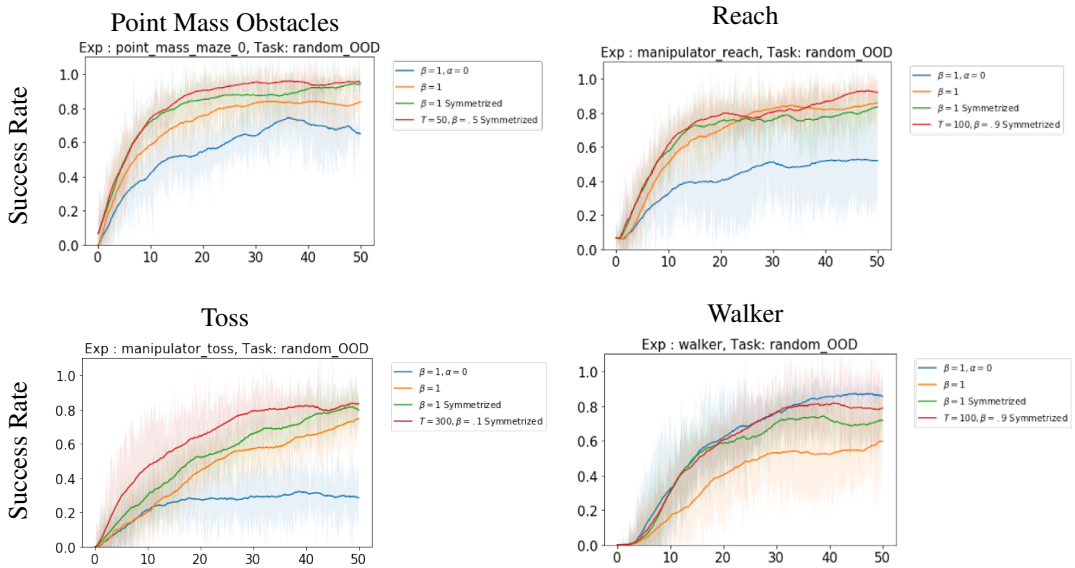


Figure 11: Success rates across environments on random goals sampled from $\mathcal{G}_{ood}$, ablating for SAC-based goal generators, symmetrization and stale regret updates. We generally find that having both components improve success rates across the tasks, although the gains vary depending on how challenging the environment is to begin with. In particular, having both the regret updates and symmetrization helps most on the more challenging Walker tasks and the skill-specific tasks.

Multiagent Motivation. Here we investigate whether it is sufficient to formulate the CuSP game with only a single learner. Rather than using regret defined as the difference in returns of two agents, we look at the difference in the returns of a single agent across two rollouts to identify if stochasticity in the policy alone is sufficient for providing the goal generator signal. As a case study we look at the Toss task in Figure 12, where we find the success rate increases much more slowly with a single agent. We hypothesize that a challenge with only using a single agent fro defining regret is that the regret landscape is much more flat – i.e. the regret is more likely to be very small and not provide much signal to the goal generators, as the stochasticity of a single agent policy is comparatively lesser than two separate agents. As a result, the goals are less likely to be a useful curriculum for the agent.

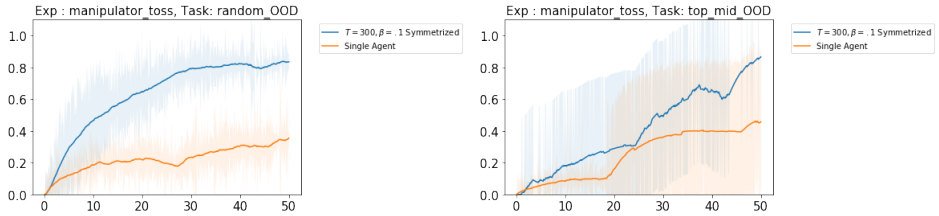
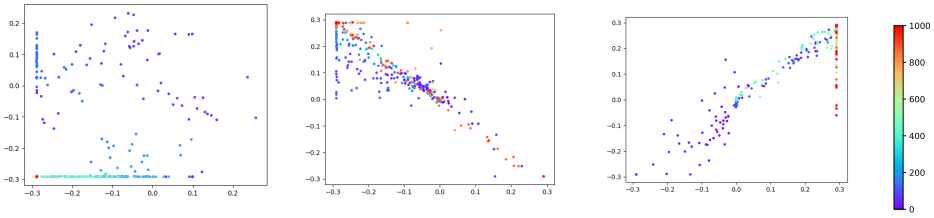


Figure 12: Comparison of CuSP with separately initialized and updated Alice and Bob, or a single Alice. Success rates are averaged across 3 seeds.

B.5.2 ASP Ablations



(a) ASP Sparse (b) ASP Dense (c) ASP Dense with SAC Bob

Figure 13: Progressive goal generation plots for the Point Mass environment after the first 1000 episodes, where each goal is the final position of Alice at the end of the episode and both agents are initialized at the center of the domain. We see that with the sparse version of ASP, the proposed goals converge on a single corner over time and gets stuck, whereas the dense implementation leads to more diverse goals.

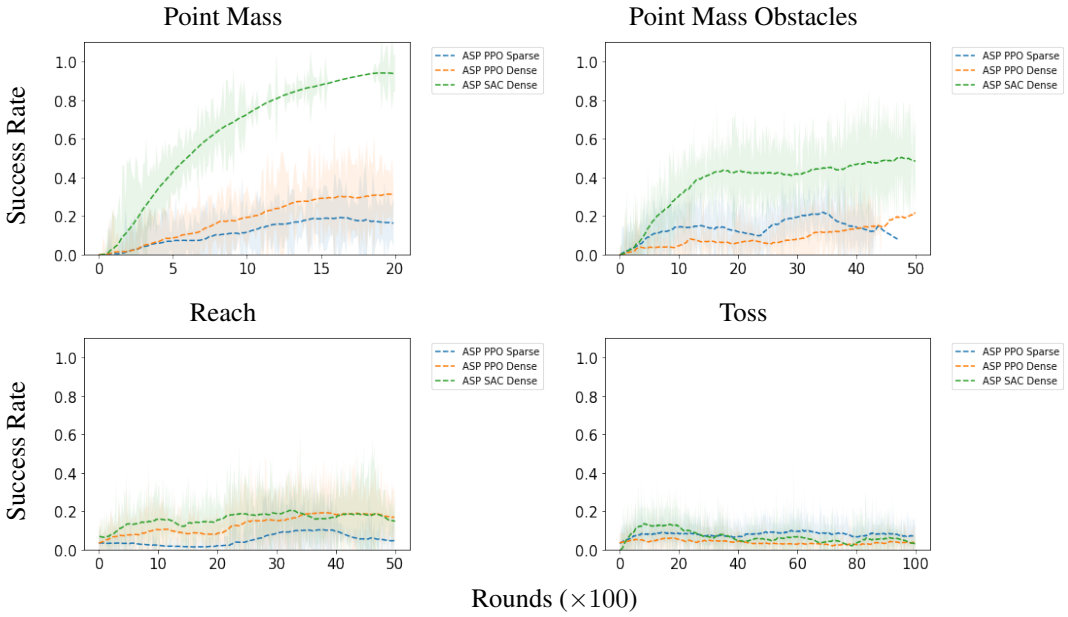


Figure 14: Success rates across environments where ASP achieved > 0 success rates on random goals sampled from $\mathcal{G}_{ood}$ across each of the ASP variants. For final results in the paper, we report ASP+BC (Dense, SAC).

In the original implementation, ASP+BC uses PPO to optimize both Alice and Bob. We find this very suboptimal for the environment setups in this work. For empirical validation, here we report results using 3 variations of ASP+BC:

- ASP+BC (Sparse): Alice is rewarded +1 at the end of an episode of Bob does not succeed, or receives 0 reward otherwise. Both Alice and Bob are optimized using PPO.
- ASP+BC (Dense): Alice is rewarded negative Bob's reward at each transition. Both Alice and Bob are optimized using PPO.
- ASP+BC (Dense, SAC): Alice is rewarded negative Bob's reward at each transition. Alice is optimized with PPO while Bob is optimized with SAC.

For all cases, we use the behavioural cloning (BC) mechanism proposed in Section 3.2 of [28], where Bob is updated with the same clipped behavioural cloning loss on trajectories where it does not successfully achieve Alice's proposed goal.

The results are shown in Figure 14. We observe that with a sparse reward, Alice would tend to get stuck with proposing similar goals in a particular location. While Bob learns to succeed at that single goal, the lack of diversity causes Bob to perform poorly at generalization to novel goals in $\mathcal{G}_{ood}$, as seen in the evaluations. To overcome this, we incorporated a dense reward implementation and swapped to optimizing for Bob's policy using SAC. Our results presented in Section 4 use this stronger baseline for comparison against CuSP.

C Theoretical Analysis

Our overall framework contains 4 players: two goal-conditioned agents, Alice π_A and Bob π_B, and their corresponding goal generators, G_A and G_B. Further, let Π and $\mathcal{G}$ denote the space of policies and goal generative models respectively. At any given round, let $g_A \sim G_A$ and $g_B \sim G_B$ denote the corresponding goals sampled from G_A and G_B respectively. Accordingly, we can define two regrets for the sampled goals:

$$\mathfrak{R}^{G_A}(g_A, \pi_A, \pi_B) = R(g_A, \pi_A) - R(g_A, \pi_B) \tag{1}$$

$$\mathfrak{R}^{G_B}(g_B, \pi_B, \pi_A) = R(g_B, \pi_B) - R(g_B, \pi_A) \tag{2}$$

where $R(g, \pi_A)$ and $R(g, \pi_B)$ denote the empirical discounted sum of rewards for Alice and Bob respectively on a goal g. Now, we define the objective function f below:

$$f(g_A, g_B, \pi_A, \pi_B) := \mathfrak{R}^{G_A}(g_A, \pi_A, \pi_B) - \mathfrak{R}^{G_B}(g_B, \pi_B, \pi_A). \tag{3}$$

The overall objective function for CuSP can be written as:

$$\min_{G_B \in \mathcal{G}, \pi_B \in \Pi} \max_{G_A \in \mathcal{G}, \pi_A \in \Pi} \mathbb{E}_{g_A \sim G_A, g_B \sim G_B}[f(g_A, g_B, \pi_A, \pi_B)] := F(G_A, G_B, \pi_A, \pi_B). \tag{4}$$

To analyze the above objective, we define *coordinating goal generator-solver teams* (G, π) as a pair of goal generator $G \in \mathcal{G}$ and goal conditioned agent $\pi \in \Pi$. Accordingly, we can interpret CuSP as a 2 player zero-sum game between two coordinating goal generator-solver teams (G_A, π_A) and (G_B, π_B) defined via goal-conditioned agents $\pi_A \in \Pi$ (Alice) and $\pi_B \in \Pi$ (Bob) along with their corresponding friendly goal generators $G_A \in \mathcal{G}$ and $G_B \in \mathcal{G}$ respectively.

Definition C.1. *(CuSP game) The CuSP game is a 2 player zero-sum game between two coordinating goal generator-solver teams (G_A, π_A) and (G_B, π_B) defined via:*

- *Alice $\pi_A \in \Pi$ and its friendly goal generator $G_A \in \mathcal{G}$ and*
- *Bob $\pi_B \in \Pi$ and its friendly goal generator $G_B \in \mathcal{G}$.*

The game is zero-sum with payoff function given as $F(G_A, G_B, \pi_A, \pi_B)$ for (G_A, π_A), and as $-F(G_A, G_B, \pi_A, \pi_B)$ for (G_B, π_B).

Next, we state conditions under which a Nash Equilibrium exists for coordinating goal generator-solver agents in CuSP. The conditions depend highly on whether the game strategies (given by the product of goal generator space and policy space) are finite or continuous.

Proposition C.2. *(Finite Games) Let $(G_A, \pi_A) \in \mathcal{G} \times \Pi$ and $(G_B, \pi_B) \in \mathcal{G} \times \Pi$ be two coordinating goal generator-solver agents for CuSP defined over finite goal space $\mathcal{G}$ and policy space Π. Then, there exists a mixed strategy Nash Equilibrium for the CuSP game.*

Proof. Since $\mathcal{G}$ and Π are finite, we know that the product space $\mathcal{G} \times \Pi$ is also finite. The result then follows directly from Nash's Theorem [25]. □

Proposition C.3. *(Continuous Games) Let $(G_A, \pi_A) \in \mathcal{G} \times \Pi$ and $(G_B, \pi_B) \in \mathcal{G} \times \Pi$ be two coordinating goal generator-solver agents for CuSP defined over continuous goal space $\mathcal{G}$ and policy space Π. Further, let $\mathcal{G}$ and Π be nonempty compact metric spaces and the payoff function F (Eq. 4) be continuous. Then, there exists a mixed strategy Nash Equilibrium for the CuSP game.*

Proof. Since $\mathcal{G}$ and Π are compact, we know by Tychonoff's theorem [43, 46] that the product space $\mathcal{G} \times \Pi$ is also compact. Combined with the fact that F is assumed to be continuous, we can apply Glicksberg's Theorem [10] to finish the proof. □

Since we have established conditions for the existence of Nash equilibria for the CuSP game, we know from the minimax theorem that we can recover the Nash equilibirum solution by optimizing Eq. 4.

Note 1: In practice, our algorithms for optimizing Eq. 4 might not satisfy the necessary assumptions for the theoretical results. For example, we optimize the parameters of generators and policies specified as neural networks as opposed to optimizing in the space of functions directly. Further, we consider a single-sample Monte Carlo estimate for the CuSP objective in Eq. 4 and optimize the goal generators and policy networks sequentially for a fixed number of gradient updates in every round using RL algorithms such as SAC. While these protocols are standard practice even in related works, deriving theoretical guarantees in such scenarios is extremely challenging and an active area of theoretical research with many open questions.

Note 2: If $\pi_A = \pi_B$, then $r_A(g) = r_B(g)$ for all g, so all corresponding regrets will also be zero and thus give no signal to the goal generators. The key point to note here is that in practice, the goal generators have an exploration component that can avoid getting stuck, e.g., this can be achieved via ϵ-greedy or via SAC which has an exploration bonus (as done in CuSP).

Concretely, let $\pi_A = \pi_B = \pi_{rand}$ be randomly initialized agents. We claim that at initialization, (G_A, π_{rand}), (G_B, π_{rand}) will not be in a Nash Equilibrium except in the case where a random policy is already able to achieve all possible goals. To see why, suppose for the sake of contradiction that the agent teams (G_A, π_{rand}), (G_B, π_{rand}) are at a Nash Equilibrium. That is, the teams have the joint best responses to each other. Let $\mathbb{G}$ be the set of all possible goals and $\mathbb{G}' \subseteq \mathbb{G}$ be the set of goals that π_{rand} can solve.

If $\mathbb{G}' = \mathbb{G}$, then π_A, π_B are already maximally competent and are therefore done learning. Note that this is also what we hope to converge to eventually during training, and the two policies matching can correspond to a Nash Equilibrium in such a case. At this point $\mathfrak{R}(g_A, \pi_A, \pi_B) = \mathfrak{R}(g_B, \pi_B, \pi_A) = 0$ for all g_A, g_B so neither G_A, G_B can increase their payoffs, and since π_A, π_B are parameterized identically, if either policy is able to increase its payoff on some g we can identically update the other policy such that $\pi_A = \pi_B$ until neither policy is able to improve its payoff further either.

If $\mathbb{G}' \subset \mathbb{G}$, then for either team we can pick any single goal $g^* \in \mathbb{G} \setminus \mathbb{G}'$ and learn a policy π^* capable of achieving $\{g^*\} \cup \mathbb{G}'$ using policy iteration (e.g., with greedy policy improvement assuming no forgetting of past goals). Such a g^* will eventually be proposed as long as the goal generators have an exploratory component. This means there is a strategy π^* that acquires equal utility to π_{rand} on $\mathbb{G}'$ and higher utility on g^*. Thus there exists a strictly better strategy for either team $(G_A, \pi_{rand}), (G_B, \pi_{rand})$ by switching from π_{rand} to π^*, which contradicts that the players are already at a Nash Equilibrium to begin with.