Risk-Aware Curriculum Generation for Heavy-Tailed Task Distributions (Supplementary Material)

Cevahir Koprulu [1] **Thiago D. Simão**[2] **Nils Jansen**[2] **Ufuk Topcu**[1]

[1]University of Texas at Austin
[2]Radboud University, Nijmegen

A AUTOMATED CURRICULUM GENERATION ALGORITHMS

This section summarizes the automated curriculum generation methods considered in the empirical evaluation.

SPDL [6]: We use *Self-paced Deep Reinforcement Learning* to generate the primary curriculum. The main document provides more details.

CURROT [7]: *Curriculum RL via Constrained Optimal Transport* formulates the curriculum generation problem as constrained optimal transport by generating a context distribution that minimizes the Wasserstein distance to the target context distribution and satisfies two constraints: the discounted return should be higher than some pre-determined threshold in every context with non-zero probability and the Wasserstein distance to the previous context distribution should be lower than some distance threshold.

PLR [4]: *Prioritized Level Replay* addresses procedural context generation environments, where a *level* is an allegorically created unique environment instance. PLR samples the next training level by prioritizing the ones with a higher average magnitude of generalized advantage estimate [10], that is the discounted sum of all temporal-difference errors occurring in the future.

VDS [11]: *Value Disagreement based Sampling* addresses the goal-conditioned setting and uses the epistemic uncertainty of the value function to sample goals. Intuitively, the value function confidently assigns low and high values to hard and easy goals, respectively, but it is uncertain about the values of the goals that are at the boundary of the current policy's ability. To generate a curriculum, VDS samples these goals for which the value function has high epistemic uncertainty.

GoalGAN [2]: *Goal Generative Adversarial Network* also addresses the goal-conditioned setting. The proposed approach uses a goal discriminator to determine whether a goal is at the intermediate difficulty for the current policy, and a goal generator that generates goals that are at such level of difficulty.

ALP-GMM [8]: *Absolute Learning Progress with Gaussian Mixture Models* generates a Gaussian mixture model over the absolute learning progress of task parameters, e.g., contexts in our setting, and uses a bandit scheme to choose a Gaussian as an arm whose utility is the absolute learning progress. The chosen Gaussian distribution is used to draw the next task parameter.

B EXPERIMENTAL DETAILS

This section discusses the hyperparameter selection process for the evaluated curriculum generation algorithms and additional details regarding the environments used in the experiments.

B.1 ALGORITHM HYPERPARAMETERS

RaCGEN, RaCGEN-N, SPDL, and SPDL-N generate curricula, primary, via the self-paced RL framework, which has four parameters: performance constraint threshold δ, KL divergence threshold ϵ, number of curriculum iterations K, and

Table 1: Self-paced RL parameter values used in RACGEN, RACGEN-N, SPDL, and SPDL-N.

Environment	δ	ϵ
PointMass-2D	4.0	0.25
LunarLander-2D	-100	0.025

Table 2: Selected values for parameters of CURROT, PLR, and VDS.

Environment	δ	ϵ_{Wass}	ρ	β	p	LR	n_{ep}	n_{batch}
PointMass-2D	4.0	0.5	0.45	0.15	0.85	0.01	5	40
LunarLander-2D	-50	0.5	0.45	0.15	0.85	0.01	3	40

number of rollouts per policy update M. For every environment, we chose δ to be around the midpoint between the minimum and maximum possible discounted return. To select KL divergence threshold ϵ, we ran a grid search over $\{0.5, 0.25, 0.1\}$ for the point-mass environment and $\{0.1, 0.05, 0.01\}$ for the lunar lander environment. We selected the values that yield the best performing RACGEN curricula, and use the same values for RACGEN-N, SPDL, and SPDL-N. For the point-mass environment, we set $K = 300$, based on the experiments of Klink et al. [6] in the same environment, and set $M = 30$ after a search over $\{30, 40\}$, similar to the previous grid search. For the lunar lander environment, we set $K = 250$, and ran a grid search for M over $\{30, 40\}$ and chose 40 based on the best performing RACGEN curriculum.

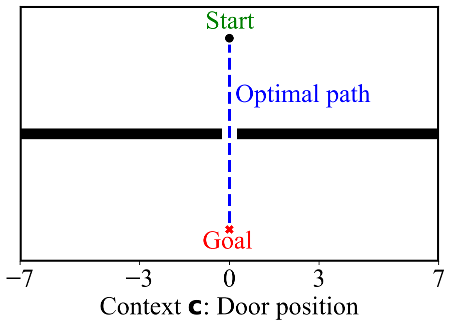
RACGEN, RACGEN-N, and DEFAULT-CEM use a CEM module to generate auxiliary curricula. There are four parameters that CEM utilizes: smoothing risk level β, final risk level α, initial risk level α_0, risk level scheduling factor ρ. We illustrate the pitfalls of heavy-tailed target context distributions based on a point-mass environment with 1D context space, where we set $\alpha = 0.2$. We use this value for both environments in the experiments. Based on the discussion in Greenberg et al. [3], we set $\beta = 0.5$ to care for the mean of the lower 50% of the samples. We set $\alpha_0 = 1$ to start soft-risk scheduling by identifying contexts that have returns lower than the expectation of the returns. We compute ρ to have a certain number of curriculum updates until α is reached. We ran grid searches over $\{10, 15, 20\}$ and $\{40, 80, 120\}$ for point-mass and lunar-lander environments, respectively. Based on the best performing RACGEN, we set ρ so that it decays to α from 1.0 in 20 and 80 curriculum updates in point-mass and lunar lander environments, respectively, for methods RACGEN, RACGEN-N, and DEFAULT-CEM. The algorithms using a CEM module also need M^{pri} and M^{aux} to be set. As $M = M^{pri} + M^{aux}$, we set M^{aux} to 20 and 10 for point-mass and lunar-lander environments, respectively.

CURROT has two main parameters: performance constraint threshold δ and Wasserstein distance threshold ϵ_{Wass}, similar to the self-paced RL algorithm we employ for RACGEN. Following the logic described by the developers of CURROT [7], we set δ to be half-way between the minimum and maximum discounted returns. In addition, ϵ_{Wass} is usually set to a high value such as 0.5, so we keep the same approach. PLR has three parameters: the staleness coefficient ρ, the score temperature β, and the replay probability p. We ran a grid search over $(\rho, \beta, p) \in \{0.15, 0.45\} \times \{0.15, 0.45\} \times \{0.7, 0.85\}$. VDS has three parameters to set: the learning rate LR for the Q-function ensemble, the number of epochs n_{ep}, and the number of minibatches n_{batch}. We ran a grid search over $(\text{LR}, n_{\text{ep}}, n_{\text{batch}}) \in \{0.0001, 0.001\} \times \{3, 5\} \times \{20, 40\}$. Table 2 consists of the final parameter values used in point-mass and lunar lander environments for CURROT, PLR, and VDS.

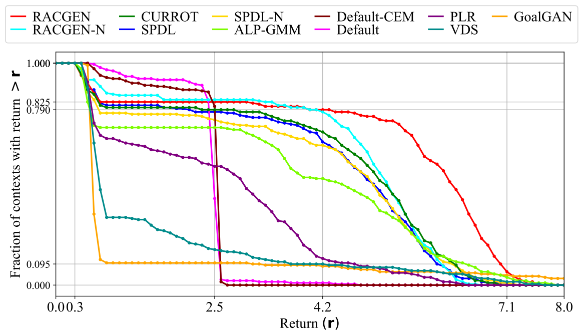
GOALGAN has three parameters: the random noise added to each context sample δ_{noise}, the number of rollouts between context distribution updates $n^{\text{GG}}_{\text{rollout}}$, and the percentage of samples drawn from the success buffer p_{success}. We ran a grid search over $(\delta_{\text{noise}}, n^{\text{GG}}_{\text{rollout}}, p_{\text{success}}) \in \{0.05, 0.1\} \times \{100, 200\} \times \{0.1, 0.2\}$. ALP-GMM has three parameters: the ratio of randomly sampled contexts p_{rand}, the number of completed learning episodes before updating the context distribution $n^{\text{AG}}_{\text{rollout}}$, and the size of the past trajectory buffer s_{buffer}. We ran a grid search over $(p_{\text{rand}}, n^{\text{AG}}_{\text{rollout}}, s_{\text{buffer}}) \in \{0.1, 0.2\} \times \{50, 100\} \times \{500, 1000\}$. Table 3 shows the final parameter used for GOALGAN, and ALP-GMM.

Table 3: Selected values for parameters of GOALGAN and ALP-GMM

Environment	δ_{noise}	$n^{\text{GG}}_{\text{rollout}}$	p_{success}	p_{rand}	$n^{\text{AG}}_{\text{rollout}}$	s_{buffer}
PointMass-2D	0.05	200	0.1	0.1	50	500
LunarLander-2D	0.1	200	0.2	0.1	100	500



(a) Visualized point-mass environment.



(b) Performance profiles in point mass environment.

Figure 1: Point-mass environment: (a) Visualization of Point-mass environment with 2D context space: Context **c** determines the position and the width of the door. (b) Performance profiles of evaluated algorithms in the point mass environment: the fraction of episodes where the final policies achieve discounted returns greater than **r**. It presents the median over 10 independent training runs.

B.2 ENVIRONMENT DESCRIPTIONS

Point-mass environment. Fig. 1b further demonstrates that RACGEN achieves higher returns in high and medium-risk contexts than the remaining methods. The figure shows the fraction of contexts (y-axis) where an algorithm learns a policy that achieves a return higher than the return $\mathbf{r}$ (x-axis). The plot shows the median over 5 runs. First, we notice that RACGEN almost always achieves returns higher than -46, with DEFAULT following closely and the rest achieving lower returns in high-risk contexts. At $\mathbf{r} = -30$, DEFAULT starts to perform worse than RACGEN, which supports our previous argument that RACGEN achieves the highest minimum returns, indeed. The curve of RACGEN stays on the top until $\mathbf{r} = 62$, which demonstrates that RACGEN performs the best in most of the contexts. However, as we discussed in Figure 6 of the main document, RACGEN does not yield the highest returns in low-risk contexts since its curve goes under the others in terms of the portion of contexts with high returns, more specifically for returns $\mathbf{r} \in [62, 74] \cup [82, 100]$.

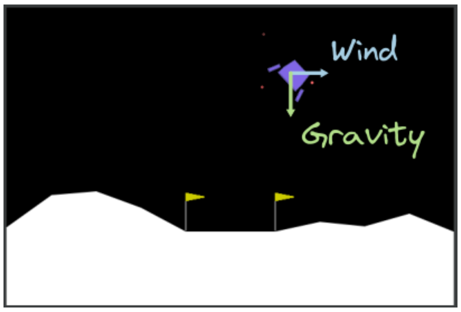
We use the environment studied by Klink et al. [5, 6, 7], which has a two-dimensional context space. A context in the point-mass environment determines the position and the width of the door, that the agent needs to pass to eventually reach the goal position. As the algorithm of choice, we employ the stable-baselines3 [9] implementation of PPO with an MLP policy of 3 hidden layers, 128 neurons at each layer. We set the batch size to 128, Generalized Advantage Estimator factor to 0.99, and the number of steps in between updates to 6144. The discount factor of the point-mass environment is 0.95. We leave the values of the rest of the parameters as set in the stable-baselines3 implementation of PPO. An illustration of the point-mass environment is in Fig. 1a.

Lunar-lander environment. The lunar-lander environment we use is the third version in OpenAI Gym [1], which has a two-dimensional context space. A context in the lunar-lander environment determines the gravity and the wind power of the planet that a pod needs to land on. We again utilize the stable-baselines3 [9] implementation of PPO with the default MLP policy. We set the number of epochs for surrogate loss optimization to 4, Generalized Advantage Estimator factor to 0.99, and the number of steps in between updates to 10240. The discount factor of the lunar-lander environment is 0.99. We leave the values of the rest of the parameters as set in the stable-baselines3 implementation of PPO. An illustration of the lunar-lander environment is in Fig. 2a.

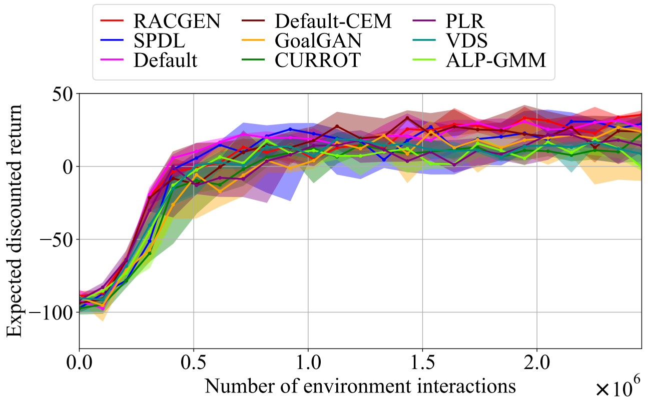
B.3 DETAILED ANALYSIS OF RESULTS

Fig. 2b shows the progression of the expected discounted return in contexts drawn from the target context distribution.

Point-mass environment. Fig. 1b demonstrates that RACGEN achieves higher returns in 79.2% of the contexts, whereas the remaining methods perform poorly in contrast. The figure shows the fraction of contexts (y-axis) where an algorithm learns a policy that achieves a return higher than the return $\mathbf{r}$ (x-axis). The curves correspond to the median over 10 runs. DEFAULT and DEFAULT-CEM mostly achieve returns around $\mathbf{r} = 2.5$, as the final policies learn to push the point mass to the middle section of the wall. Such behavior is suboptimal when the task is to pass doors away from the middle section. In



(a) Visualized lunar-lander.



(b) Expected discounted return progression in lunar lander.

Figure 2: Lunar-lander environment: (a) Visualization of the environment, where Context $\mathbf{c}$ determines the wind and the gravity. (b) Expected discounted return with respect to the target context distribution in the lunar-lander environment. The bold lines are the median and the lightly shaded regions cover the first and third quartiles of 5 independent training runs.

Table 4: Statistics of distributions of discounted returns collected by policies trained under listed algorithms in the lunar-lander environment. We use the discounted returns collected in 100 contexts (drawn from the target context distribution) by policies from 5 independent runs (in total 500 discounted returns per algorithm)

Algorithms	Maximum (Upper Whisker)	3rd Quartile	Median	1st Quartile	Minimum (Lower Whisker)
RACGEN	88.57	**48.89**	**31.74**	**15.77**	**-30.25**
DEFAULT-CEM	**101.45**	48.22	30.77	11.22	-44.11
DEFAULT	92.08	43.86	27.45	11.37	-34.29
SPDL	90.13	44.22	25.02	-1.45	-67.89
CURROT	79.79	36.67	22.29	7.94	-33.52
PLR	74.91	30.76	15.29	-0.81	-47.93
GOALGAN	93.51	35.20	15.19	-7.67	-68.93
VDS	90.00	27.87	8.06	-17.44	-85.32
ALP-GMM	90.07	24.68	5.51	-20.32	-80.70

comparison, RACGEN and RACGEN-N yield lower returns in 20% of the contexts, as the learned policies sometimes approach the door but fail to pass it. The curve of RACGEN stays on top of all evaluated methods for $\mathbf{r} \in [4.2, 7.2]$, as RACGEN generates Cauchy context distributions instead of Gaussian when the target context distribution is Cauchy. Although GOALGAN achieves higher returns than RACGEN in less than 10% of the contexts, the figure for the distribution of returns in the main document indicates that such cases are outliers.

Lunar-lander environment. Table 4 consists of the numerical values from the boxplots in Figure 6. As we indicate in Section 6.2, RACGEN outperforms all algorithms regarding median, the first and third quartiles, and minimum values. However, DEFAULT-CEM is the best performer considering the maximum return achieved. As RACGEN attends risky contexts much more than easy ones, it may have overlooked high-return contexts more than the baselines.

In Table 4, we also observe that RACGEN has a tighter range and less spread-out low outliers than DEFAULT-CEM. Even though both approaches identify and oversample rare and risky contexts using their CEM modules, RACGEN performs better in such contexts. Furthermore, DEFAULT (without a curriculum and a CEM module) yields higher first quartile and minimum values than DEFAULT-CEM. Thus, we argue that RACGEN is more robust than the baselines.

Table 5 provides the median returns in contexts, namely, the median in a context across independent training runs. As we focus on the median returns, we disregard training runs that yield unusually high or low returns in a context. In this case, RACGEN outperforms all algorithms in every statistic. Therefore, Table 5 also supports our argument that RACGEN is more advantageous than all state-of-the-art algorithms and baselines in the lunar lander experiment.

Table 5: Statistics of distributions of discounted returns collected by policies trained under listed algorithms in the lunar-lander environment, where we focus on the median return across 5 independent runs in 100 contexts drawn from the target context distribution.

Algorithms	Maximum (Upper Whisker)	3rd Quartile	Median	1st Quartile	Minimum (Lower Whisker)
RACGEN	**69.53**	**42.79**	**32.42**	**23.39**	**0.40**
DEFAULT-CEM	66.03	39.86	29.89	18.78	-5.74
DEFAULT	56.14	35.03	27.72	20.49	-0.48
SPDL	64.18	38.16	27.45	15.16	-17.56
CURROT	52.07	30.37	22.01	15.13	-7.23
PLR	41.47	24.13	16.29	8.15	-11.97
GOALGAN	48.23	24.21	14.01	5.88	-12.67
VDS	44.06	20.97	8.14	-4.88	-41.83
ALP-GMM	42.04	17.19	6.23	-5.17	-36.37

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic Goal Generation for Reinforcement Learning Agents. In *ICML*, pages 1514–1523. PMLR, 2018.

[3] Ido Greenberg, Yinlam Chow, Mohammad Ghavamzadeh, and Shie Mannor. Efficient Risk-Averse Reinforcement Learning. In *NeurIPS*, 2022.

[4] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *ICML*, pages 4940–4950. PMLR, 2021.

[5] Pascal Klink, Carlo D' Eramo, Jan R Peters, and Joni Pajarinen. Self-paced deep reinforcement learning. In *NeurIPS*, pages 9216–9227. Curran Associates, Inc., 2020.

[6] Pascal Klink, Hany Abdulsamad, Boris Belousov, Carlo D'Eramo, Jan Peters, and Joni Pajarinen. A probabilistic interpretation of self-paced learning with applications to reinforcement learning. *JMLR*, 22:182:1–182:52, 2021.

[7] Pascal Klink, Haoyi Yang, Carlo D'Eramo, Jan Peters, and Joni Pajarinen. Curriculum reinforcement learning via constrained optimal transport. In *ICML*, pages 11341–11358. PMLR, 2022.

[8] Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. In *CoRL*, pages 835–853. PMLR, 2020.

[9] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *JMLR*, 22(268):1–8, 2021.

[10] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

[11] Yunzhi Zhang, Pieter Abbeel, and Lerrel Pinto. Automatic curriculum learning through value disagreement. In *NeurIPS*, pages 7648–7659. Curran Associates, Inc., 2020.