Towards Self-Paced Context Evaluation for Contextual Reinforcement Learning

Theresa Eimer [1] **André Biedenkapp** [2] **Frank Hutter** [2 3] **Marius Lindauer** [1]

Abstract

Reinforcement Learning has performed very well on games and lab-based tasks. However, learning policies across a distribution of instances of the same task still remains challenging. Recent approaches assume either little variation between instances or an unlimited amount of training examples from a given distribution. Both properties are not always feasible in real-world applications. Thus, we need methods that enable agents to generalize from a limited set of example instances or experiences. We present an approach, based on self-paced learning, that allows to exploit the information contained in state values during training to accelerate and improve training performance as well as generalization capabilities, independent of the problem domain at hand. The proposed *Self-Paced Context Evaluation (SPaCE)* provides a way to automatically generate instance curricula online with little computational overhead.

1. Introduction

Progress in reinforcement learning (RL) so far has largely been made in artificial settings with ideal conditions, such as unlimited environment interactions (Silver et al., 2016) or availability of a target distribution to learn (Abbeel & Ng, 2004). When intending to apply RL in real-world domains, such as healthcare or autonomous driving, it is unrealistic to expect these idealized settings. An RL agent will have to be able to handle both larger variations in instance distributions for the same task and, at the same time, limited access to samples from the same distribution. This could be achieved either through offline RL with previously collected experiences (Fu et al., 2020) or online by achieving the best performance possible on a limited set of instances from the same distribution; the latter is our focus here. In these cases, generalization is an even greater challenge than in an idealized lab setting, requiring a well thought-out strategy to maximize learning from our limited data.

A method that has been shown to both accelerate learning but also improve generalization performance in RL is Curriculum Learning (CL; Bengio et al., 2009; Florensa et al., 2018; Wang et al., 2019). Similar to the way humans learn, CL aims to exploit knowledge gained from easy instances to solve harder ones. In order to do this, instead of training on instances in a round robin fashion, i.e., iterating over all instances in an arbitrarily fixed order, the agent starts training on easy ones while the difficulty is increased over time. Manually finding the best instance ordering for an agent, however, requires expert knowledge about both the task and the RL algorithm. The problem is further complicated by the fact that the timing at which new instances should be added also plays a crucial role. Adding easy instances too slowly will not give the agent the opportunity to leverage the knowledge from previous tasks, while adding difficult instances to quickly may not allow it to learn at all.

Although various curriculum learning methods for RL have emerged, they assume unlimited access to a distribution from which to sample instances (OpenAI et al., 2019; Klink et al., 2019; 2020). Generally, they propose that choosing a narrower slice of this distribution makes learning the task easier, thus beginning training by sampling from a subset of the final instance distribution and widening it as the agent improves. Such methods, while effective, are not directly applicable to many potential applications of RL without a generator to sample from such a distribution.

In this work, we present an automatic curriculum learning method for contextual RL that requires neither domain knowledge nor access to a distribution from which one can sample. Our method, *Self-Paced Context Evaluation (SPaCE)*, evaluates the difficulty of given instances instead of sampling instances to match the current difficulty level. To assess instance difficulty efficiently and accurately, we use an agent's own knowledge, contained in the value function. Thus, we can use any RL agent with a value-function, including value-based agents and policy gradient agents, to

[1]Information Processing Institute (tnt), Leibniz University Hannover, Germany [2]Department of Computer Science, University of Freiburg, Germany [3]Bosch Center for Artificial Intelligence, Renningen, Germany. Correspondence to: Theresa Eimer <eimer@tnt.uni-hannover.de>.

Preliminary work. Big@ICML Workshop, 2020.

score the current subjective difficulty of training instances with a single forward pass over the training set. These difficulty estimations let us order the instances in the training set freely according to the agent's current performance. We show that SPaCE is capable of outperforming round robin as a default instance ordering on the contextual PointMass environment (Klink et al., 2020) both during training and in generalization to an unseen test set.

2. Contextual Reinforcement Learning

We model RL across different instances of the same task as a contextual Markov Decision Process (cMDP). An *instance* here could mean a different goal position in a maze task or a different object for a robot hand to pick up. Let $\mathcal{I}$ be a set of instances i, then the contextual MDP $\mathcal{M}_\mathcal{I}$ is a collection of MDPs $\mathcal{M}_\mathcal{I} \coloneqq \{\mathcal{M}_i\}_{i\in\mathcal{I}}$ with $\mathcal{M}_i \coloneqq (S, A, T_i, R_i)$ (Hallak et al., 2015). As the underlying task stays the same, we assume the state and action spaces (S and A, respectively) are consistent across all instances; however, the transition and reward functions (T_i and R_i, respectively) are unique to each instance.[1]

An optimal policy π^* for this cMDP optimizes the expected return over all $i \in \mathcal{I}$ (Klink et al., 2020):

$$\pi^* \in \arg\max_{\pi\in\Pi} \frac{1}{|\mathcal{I}|} \sum_{i\in\mathcal{I}} \mathbb{E}_{\pi,t} \gamma^t \mathcal{R}_i^\pi(s_t). \tag{1}$$

As the expectation depends on the given instance, an agent solving the cMDP needs to be given the instance context in order to differentiate between instances. This will be indicated as c_i for instance $i \in \mathcal{I}$.

3. Related Work

POET (Wang et al., 2019) and ADR (OpenAI et al., 2019) are two methods that use CL to advance the instance difficulty for a given task towards open-endedness. POET surpassed what conventionally trained agents could achieve by evolving instances in parallel s.t. their difficulty continually increased to match the agent's capabilities. Newly sampled instances were assessed using domain knowledge and resampled if needed. ADR, meanwhile, was used to solve a Rubik's cube with a robot hand from more and more difficult positions. The cube positions are sampled from an expanding distribution that is updated whenever the agent has solved a sufficient number of cubes. In contrast to this line of work, we assume to have access only to a limited number of example instances from which we can learn.

Similarly to ADR, Goal GAN aims to provide the agent with instances of increasing difficulty, but uses a generative adversarial network to generate these instances (Florensa et al., 2018). In this setting, an agent acts as discriminator, indicating by its success ratio whether generated samples are of the desired difficulty. As generator and discriminator are trained jointly, this automatically generates a curriculum.

While these methods are conscious of an agent's progress, Self-Paced Learning (SPL; Kumar et al., 2010) suggests to explicitly use the agent's performance to construct the curriculum, allowing it to learn at its own pace. Exhaustively evaluating the agent on all available instances would be inefficient for RL, so self-paced RL methods usually use a state value estimation as a stand in. Self-Paced Contextual Reinforcement Learning (Klink et al., 2019) extends an instance sampling distribution, as done in ADR, but the extension is paced according to the agent's performance estimate. Self-Paced Deep Reinforcement Learning (SPDRL; Klink et al., 2020) extends upon this idea and shows self-paced learning to outperform other CL methods like the above mentioned goal GAN. This is similar to our proposed method, but still assumes that we know the underlying target instance distribution.

ALLSTEPS (Xie et al., 2020) uses SPL in a different way, not to sample new instances but to extend a continuous path for human walking training. By continuously simulating and evaluating possible next stepping positions, they create open-ended paths of increasing difficulty.

4. Self-Paced Context Evaluation

In order to generate a curriculum without any prior knowledge of the target domain, we take advantage of the information contained in the agent's state value predictions. Over time, $V(s_t, c_i)$ converges towards the maximum expected reward gained from state s_t in instance i when following the current policy π (Sutton & Barto, 1998). Therefore, we propose $V(s_0, c_i)$ as an estimate of the total expected reward given s_0 of any $i \in \mathcal{I}$ and define instance difficulty wrt π and $\forall i \in \mathcal{I}$ as $d(i) = V(s_0, c_i)$. As most state-of-the-art RL algorithms use a value-based critic, this difficulty estimation is easily computed during training.

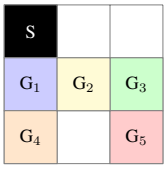
The difficulty metric is used to compile progressively larger, more difficult training sets every time the agent starts to converge on the previous one, as it has learned a suitable policy on the simpler instances. We measure this in terms of the difference in value predictions of the training set compared to the last training iteration. Once the absolute change in state values of the current training set falls below a predefined percentage η between iterations, the size of the training set is increased and its instances are chosen according to their new difficulty estimation (see Algorithm 1). η is a hyperparameter of SPaCE, as is the step size κ at which we increase the instance set. The size of the training set is

[1] For simplicity, we assume s_0 to be the starting state of the cMDP and independent of the context. Nevertheless, SPaCE can be extended to context-specific distributions over possible s_0.

Algorithm 1: SPaCE curriculum generation

Input : Agent with policy π and value function V_0,
Instance set $\mathcal{I}$, performance threshold η,
instance set step size κ, number of episodes N

$size = 0$
$\Delta V = 0$
for $n \leq N$ **do**
  if $\Delta V < \eta \cdot \sum_{i \in I_{curr}} |V_n(s_0, c_i)|$ **then**
    $size += \kappa$
  forall $i \in \mathcal{I}$ **do**
    predictions[i] $\leftarrow V_{n(s_0,c_i)}$
  $d_n = sort\_descending(\text{predictions})[:size]$
  $\mathcal{I}_{curr} := \{i \in \mathcal{I} \mid V_n(s_0, c_i) \in d_n\}$
  forall $i \in I_{curr}$ **do**
    Train π on i and update V_n
  $\Delta V := \sum_{i \in I_{curr}} |V_n(s_0, c_i) - V_{n-1}(s_0, c_i)|$



(a) Gridworld

	G1	G2	G3	G4	G5
0	2.19	3.64	5.83	4.37	2.92
1-4	2.19	2.19	2.19	2.19	2.19
5	2.19	2.19	2.19	2.19	2.19
6-9	3.64	2.19	3.64	2.19	3.64
10	3.64	2.19	3.64	2.19	3.64
11	3.64	5.83	2.19	3.64	5.83
12+	3.64	5.83	4.37	2.19	3.64

(b) SPaCE Curriculum

Figure 1: (a) Gridworld with instance start position (S) and goals of all five instances (G_i). (b) Corresponding curriculum. Rows indicate training iterations, columns state evaluations. Colored cells comprise I_{curr}.

extended instead of simply shifting it to harder instances in order to prevent forgetting previously learnt policies.

As an example, we consider a set of 5 Gridworld instances, each set in a 3x3 grid, but with differing goal positions (see Figure 1a). The state description contains an agent's current position and the number of steps the agent has taken so far. The context information is simply the location of the goal state. A reward of -1 will be given for each step until it reaches the goal state. The agent can move in all 4 directions. It will always start in the upper left corner and we allow for at most 10 steps. We use a small DoubleDQN (van Hasselt et al., 2016; Fujita et al., 2019) (see Appendix A.1) with $V(s_t, c_i) = max_a Q(s_t, c_i, a)$. We set $\kappa = 1$ to add one instance at a time when the difference in value estimation falls below $\eta = 10\%$. Figure 1b shows how SPaCE constructs the curriculum in the first few training episodes. As the self-paced agent's value estimation is initialized randomly, its initial difficulty estimations are incorrect. Thus, this agent actually does not start training on the easiest instance. After 5 iterations on single instances, it can still transfer its performance onto the other instances easily, however, as the training set is then extended to 2 instances until iteration 9 to and shortly after the complete instance set after iteration 12. The performance reflects this, as the agent is very quickly close to the optimal reward of -0.2 (see Figure 2) when using SPaCE because the single instances are very easy and the agent can transfer at least parts of its policy between them. Clearly the agent is able learn the single instances quickly, as shown by the SPaCE trained agent, but iterating over the instances in a round robin fashion makes it harder for the agent to learn. This is demonstrated as the round robin agent does not converge and is much less stable between runs.

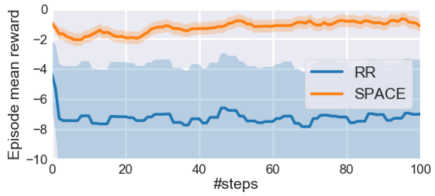


Figure 2: Mean train performance over 5 training episodes on Gridworld for round robin and SPaCE. Result shown is the mean over 10 random seeds with standard deviation.

5. Preliminary Experiments

We evaluated our approach on the contextual PointMass environment described by Klink et al. (2020) as we are using a similar approach to instance difficulty estimation.[2] In this environment, the agent maneuvers a point mass through a goal in a two-dimensional space. The goal position, width as well as the friction coefficient of the ground are given as context and together make up the instance description.

We sampled train and test sets of 100 instances each from a uniform distribution and evaluated the agent over 10 random seeds to account for randomness. We train a PPO agent (Schulman et al., 2017; Liang et al., 2018) and base our curriculum generation on its value-based actor (see A.2). For easier readability, all plots are smoothed over 10 iterations.

Figure 3 shows the comparison between SPaCE and round robin, as a baseline, during training. While round robin struggles to improve, SPaCE is quickly able to generate a reward of about 6 out of 10 points per episode. In contrast to this, round robin stays below a reward of 4 points. After each training iteration, we evaluated the agent on the test set in order to monitor generalization progress over time.

[2]Code available at: https://github.com/automl/SPaCE

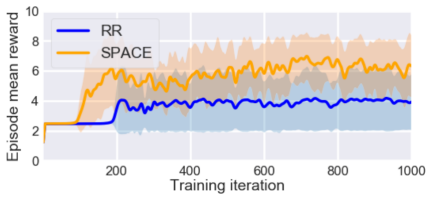


Figure 3: Mean reward per training iteration over 5 runs each with standard deviation

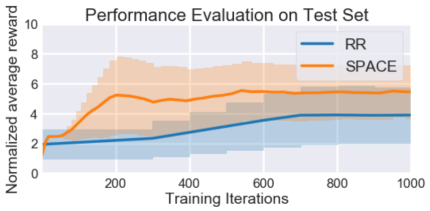


Figure 4: Mean reward per episode on test set.

The results on the test set (see Figure 4) are similar to the performance during training, with SPaCE outperforming round robin by quite a large margin of at least 2 to 4 points at all times. Obviously this agent, while being able to improve at least a little bit during training, was not able to transfer those improvements to the instance set as a whole on any run. The SPaCE agent also does not reach the optimal performance of 10, but comes much closer on both training and test set. Furthermore, the performance between training and test set does not differ very much for this agent. This indicates that our agent actually succeeded in learning the underlying instance distribution from our training set.

Considerations for Comparing SPDRL and SPaCE. As we used the same environment and similar methods as (Klink et al., 2020), we would like to comment on the differences between our SPaCE and their SPDRL. We used a PPO agent for training, while SPDRL used TRPO. Both algorithms, however, have been shown to behave very similarly (Engstrom et al., 2020), nevertheless a possible confounding factor. Our agent achieved its final performance range very fast, hovering around 6 reward points after already 200 iterations. The SPDRL-TRPO agent took longer, about 400 iterations to reach the same reward and then continued to improve to about 9 reward points by iteration 1 000. All other methods they compared against stayed below 6 reward points for the duration of training.

It is to be expected, that access to a distribution from which to smoothly sample instances, as is the case in SPDRL, would improve overall performance. We can see the limitations of our fixed-size instance set in our agent's test performance on individual instances. While the curriculum extends the instance set to the whole training set within 250

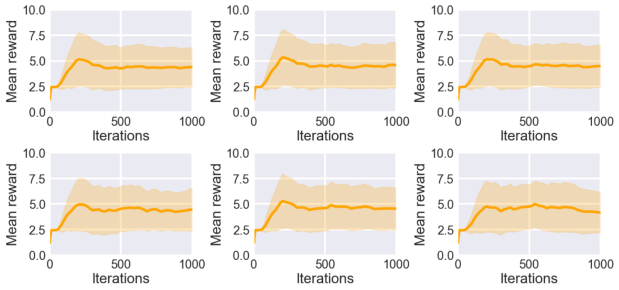


Figure 5: Examples of test instances our agent struggles to generalize to.

iterations, there are test instances on which the agent never performs well (see Figure 5) and even deteriorates over time. For our agent to be able to perform as well as the SPDRL agent, we would also have to be able to generalize to these instances. As there is an initial performance spike, it does not seem like these instances are inherently unsolvable for our agent. More likely, they are simply not represented well in the training set, see Appendix B. This is supported by the performance decreasing over time (the agent forgetting better policies for underrepresented instances in favor of improving on the remaining instances). Therefore the stagnating performance on the test set as a whole for the later half of training can be attributed at least in part to the limitations imposed on us by our training set of only 100 instances. As limited training data availability is a major consideration for many practical applications, we plan on further experiments that better quantify how much generalization is influenced by different sized training sets compared to unlimited instance sampling.

6. Conclusion

Self-Paced Context Evaluation (SPaCE) provides an adaptive curriculum learning method for problem settings constrained to a fixed set of training instances. Therefore we can transfer the generalization improvements made by simulation based self-paced learning approaches with an infinite task distribution more broadly in practice. We have shown that SPaCe outperforms round robin trained agents during training and improves the capability to generalize over the underlying instance distribution even without having direct access to it. Further research could answer how an increase in instance set size can mitigate the performance gap to sample-based approaches and if we can derive performance expectations for practical applications of RL with a limited amount of instances wrt the amount of information available. Furthermore, we might be able to use the value estimation to further improve the training efficiency, for example by clustering instances of similar difficulty and then limiting the amount of training on very easy ones to a minimum.

Acknowledgements

We thank the reviewers for their insightful feedback. The authors acknowledge funding by the Robert Bosch GmbH.

References

Abbeel, P. and Ng, A. Y. Apprenticeship learning via inverse reinforcement learning. In Greiner, R. (ed.), *Proceedings of the 21st International Conference on Machine Learning (ICML'04)*. Omnipress, 2004.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. Curriculum learning. In Bottou, L. and Littman, M. (eds.), *Proceedings of the 26th International Conference on Machine Learning (ICML'09)*, pp. 41–48. Omnipress, 2009.

Engstrom, L., Ilyas, A., Santurkar, S., Tsipras, D., Janoos, F., Rudolph, L., and Madry, A. Implementation matters in deep rl: A case study on ppo and trpo. In *Proceedings of the International Conference on Learning Representations (ICLR'20)*, 2020. Published online: `iclr.cc`.

Florensa, C., Held, D., Geng, X., and Abbeel, P. Automatic goal generation for reinforcement learning agents. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, pp. 1514–1523. Proceedings of Machine Learning Research, 2018.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv:2004.07219 [cs.LG]*, 2020.

Fujita, Y., Kataoka, T., Nagarajan, P., and Ishikawa, T. ChainerRL: A deep reinforcement learning library. In *Workshop on Deep Reinforcement Learning at the 33rd Conference on Neural Information Processing Systems (NeurIPS)'19*, December 2019.

Hallak, A., Castro, D. D., and Mannor, S. Contextual markov decision processes. *arXiv:1502.02259 [stat.ML]*, 2015.

Klink, P., Abdulsamad, H., Belousov, B., and Peters, J. Self-paced contextual reinforcement learning. In *Proceedings of the 3rd Annual Conference on Robot Learning (CoRL)*, volume 100, pp. 513–529. PMLR, 2019.

Klink, P., D'Eramo, C., Peters, J., and Pajarinen, J. Self-paced deep reinforcement learning. *arXiv:2004.11812 [cs.LG]*, 2020.

Kumar, M. P., Packer, B., and Koller, D. Self-paced learning for latent variable models. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A. (eds.), *Proceedings of the 24th International Conference on Advances in Neural Information Processing Systems (NeurIPS'10)*, pp. 1189–1197, 2010.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M. I., and Stoica, I. RLlib: Abstractions for distributed reinforcement learning. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning (ICML'18)*, volume 80, pp. 3059–3068. Proceedings of Machine Learning Research, 2018.

OpenAI, Akkaya, I., Andrychowicz, M., Chociej, M., Litwin, M., McGrew, B., Petron, A., Paino, A., Plappert, M., Powell, G., Ribas, R., Schneider, J., Tezak, N., Tworek, J., Welinder, P., Weng, L., Yuan, Q., Zaremba, W., and Zhang, L. Solving rubik's cube with a robot hand. *arXiv:1910.07113 [cs.LG]*, 2019.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv:1707.06347 [cs.LG]*, 2017.

Silver, D., Huang, A., Maddison, C., Guez, A., Sifre, L., Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning - an Introduction*. Adaptive computation and machine learning. MIT Press, 1998.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In Schuurmans, D. and Wellman, M. (eds.), *Proceedings of the Thirtieth National Conference on Artificial Intelligence (AAAI'16)*, pp. 2094–2100. AAAI Press, 2016.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. Paired open-ended trailblazer (POET): endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv:1901.01753 [cs.NE]*, 2019.

Xie, Z., Ling, H. Y., Kim, N. H., and van de Panne, M. ALLSTEPS: Curriculum-driven learning of stepping stone skills. *arXiv:2005.04323 [cs.GR]*, 2020.

A. Algorithms & Hyperparameters

A.1. Gridworld

We used the ChainerRL DoubleDQN agent with a fully connected discrete Q-function using one layer and 32 units. As an optimizer we used Adam with a learning rate of 0.001 and a constant $\epsilon = 0.1$ for exploration. We used a discount factor of $\gamma = 0.99$. The agent had access to a prioritized replay buffer with a capacity of $5 \cdot 10^5$ (much larger than the number of training steps). As this is a minimal example with very easy instances, we kept the minibatch size at 1 to be able to see the results of every update. Q-values were retrieved via the agent's statistics.

A.2. PointMass RL model

Training for PointMass was done with Ray's `PPOTrainer` class and a wrapper for the environment to change to the next training instance upon reset. We used the pytorch version of the default model with the value-based critic option enabled. To compute the average change in the value function, after each iteration we called the model with the initial observations for each instance in the current training set and then queried the value function output. Difficulty estimation was done in the same way, but for all training instances.

B. PointMass Instance distribution

As mentioned in Section 5, we assume overall test performance would improve if we had access to unlimited instances. To support this, we show the train and test instance set we used to train our SPaCE agent on the PointMass environment in Figure 6. We see that all performances on which the agent perform bad in testing are low friction instances and on the fringes of our instance space.

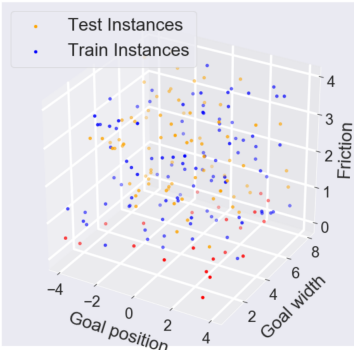


Figure 6: Training and test instance distribution. Test instances with mean performance of less than 4.5 are marked red.

Figure 7 shows the friction specifically in relation to performance. We can see that not only all instances with low test performance are instances with a low friction coefficient, all instances with low friction also have bad test performance. As we have seen improvements on these instances during training, we think this supports the idea that the lack of more low friction instances in the training set made performance improvement on these instances much harder to achieve. This is another point we want to pursue in future experiments.

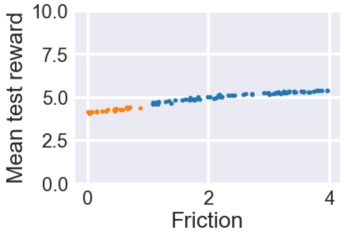


Figure 7: Mean test performance over the whole training duration in relation to instance friction. Highlighted instances have mean test performance of lower than 4.5

C. Rewards on Different Scales

An implicit assumption about our method is that the reward across instances is on a similar scale. The reason for this is that we use the expected reward as a measure of difficulty. There are reward functions for which this is not the case, however, an a common example being the reward depending on the length of the solution (e.g. maze tasks). If the objective is to finish a task very quickly but the context can significantly prolong the task without impacting the policy, it is possible for two instances to have a very similar policy but their different reward scales will prevent them from being added to the curriculum around the same time. Therefore the agent may not be able to exploit its knowledge optimally and the curriculum is weaker for it. If the length of the optimal policy is known, we could still scale the rewards, of course. However, such prior knowledge is not always available s.t. an a-priori scaling might not be possible.

Assuming similar instances with different reward scales exist, the curriculum would likely become less effective. Knowledge transfer will still happen, but with more instances having evaluations that are hard to compare to others, the quality of the curriculum will be lower. The size of this impact depends on the specific reward function and given instance set. This assumption is true for all self-paced methods and the solution is still an open question for future work.