Long-term exploration in persistent MDPs

Leonid Ugadiarov[1], Alexey Skrynnik[1,2], and Aleksandr I.
Panov[1,2][0000−0002−9747−3837]

[1] Moscow Institute of Physics and Technology, Moscow, Russia
`panov.ai@mipt.ru`
[2] Artificial Intelligence Research Institute FRC CSC RAS, Moscow, Russia

Abstract. Exploration is an essential part of reinforcement learning,
which restricts the quality of learned policy. Hard-exploration environments are defined by huge state space and sparse rewards. In such conditions, an exhaustive exploration of the environment is often impossible, and the successful training of an agent requires a lot of interaction steps. In this paper, we propose an exploration method Rollback-Explore (RbExplore) that utilizes the concept of the persistent Markov
decision process, in which agents during training can roll back to visited states. We test our algorithm in the hard-exploration Prince of
Persia game, without rewards and domain knowledge. In all used levels of the game, our agent outperforms or shows comparable results with
state-of-the-art curiosity methods with knowledge-based intrinsic motivation: ICM and RND. An implementation of RbExplore can be found
at `https://github.com/cds-mipt/RbExplore`.

Keywords: Reinforcement learning · Curiosity based exploration · State
space clustering

1 Introduction

Exploration is an essential component of reinforcement learning (RL). During
training, agents have to choose between exploiting the current policy and exploring the environment. On the one hand, exploration can make the training process
more efficient and can improve the current policy. On the other hand, excessive
exploration may waste computing resources visiting task-irrelevant regions of
the environment [4] [6].

Exploration is essential to solving sparse-reward tasks in environments with
high dimensional state space. In this case, an exhaustive exploration of the environment is impossible in practice. A considerable amount of interaction data
is required to train an effective policy due to the sparseness of the reward. A
common approach is to use knowledge-based or competence-based intrinsic motivation [10]. In the first more commonly used approach it is proposed to augment
extrinsic reward with the additional dense intrinsic reward that encourages exploration [2,3,15]. Another approach is to separate an exploration phase from a
learning phase [6]. As noted by the authors of [6], the disadvantage of the first
approach is that intrinsic reward is a non-renewable resource. After exploring

an area and consuming intrinsic reward, the agent likely will never return to the area to continue exploration due to catastrophic forgetting and inability to rediscover the path because it has already consumed the intrinsic reward that could lead to the area.

Implementing a mechanism that reliably returns the agent to the neighborhood of known states from which further exploration might be most effective is a challenging task for both approaches. In the case of resettable environments (e.g., Atari games or some robotic simulators), it is possible to save the current state of the simulator and restore it in the future. Many real-world RL applications are inherently reset-free and require a non-episodic learning process. Examples of this class of problems include robotics problems in real-world settings and problems in domains where effective simulators are not available and agents have to learn directly in the real world. Recent work has focused on reset-free setting [17,18]. On the other hand, for many domains, simulators are available and widely used at least in the pretraining phase (e.g. robotics simulators [9]). Specific properties of resettable environments make it possible to reliably return to previously visited states and increase exploration efficiency by reducing the required number of interactions with the environment. Therefore, exploration algorithms should effectively visit all states of an environment. However, taking into account the high dimension of the state space, it is intractable in practice to store all the visited states. Therefore, effective exploration of the environment remains a difficult problem, even for resettable environments.

In this paper, we propose to formalize the interaction with resettable environments as a persistent Markov decision process (pMDP). We introduce the RbExplore algorithm, which combines the properties of pMDP with clustering of the state space based on similarity of states to approach long-term exploration problems. The distance between states in trajectories is used as a feature for clustering. The states located close to each other are considered similar. The states distant from each other are considered dissimilar. Clusters are organized into a directed graph where vertices correspond to clusters, and arcs correspond to possible transitions between states belonging to different clusters. RbExplore uses a novelty detection module as a filter of perspective states. We introduce the Prince of Persia game environment as a hard-exploration benchmark suitable for comparing various exploration methods. The percentage coverage metric of the game's levels is proposed to evaluate exploration. RbExplore outperforms or shows comparable performance with state-of-the-art curiosity methods ICM and RND on different levels of The Prince of Persia environment.

2   Related Work

Three types of exploration policies can be indicated. Exploration policies of the first type use intrinsic reward as an exploration bonus. Exploration strategies of the second type are specific to multi-goal RL settings where exploration is driven by selecting sub-goals. Exploration policy of the third type use clustered representation of the set of visited states.

In recent works [4,3,11,15], the a curiosity-driven exploration of deep RL agents is investigated. The exploration methods proposed by these works can be attributed to the first type. The extrinsic sparse reward is replaced or augmented by a dense intrinsic reward measuring the curiosity or uncertainty of the agent at a given state. In this way, the agent is encouraged to explore unseen scenarios and unexplored regions of the environment. It has been shown that such a curiosity-driven policy can improve learning efficiency, overcome the sparse reward problem to some extent, and successfully learn challenging tasks in no-reward settings.

Another line of recent work focuses on multi-goal RL and can be attributed to the second type. Algorithm HER [1] augments trajectories in the memory buffer by replacing the original goals with the actually achieved goals. It allows to get a positive reward for the initially unsuccessful experience, makes reward signal denser, and learning more efficient especially in sparse-reward environments. A number of RL methods [13,7] focus on developing a better policy for selecting sub-goals for augmentation of failure trajectories in order to improve HER. These policies ensure that the distribution of the selected goals adaptively changes throughout training. The distribution should have greater variance in the early stages of training and direct the agent to the original goal in the latter stages. Other works [8,12,16] propose methods to generate goals that are feasible and their complexity corresponds to the quality of the agent's policy. The distribution of generated goals adaptively changes to support sufficient variance ensuring exploration in goal space.

Go-Explore [6] algorithm could be attributed to the third type of exploration policy. It builds a clustered lower-dimensional representation of a set of visited states in the form of an archive of cells. Two types of representation are proposed for Montezuma's Revenge environment: with domain knowledge based on discretized agent coordinates, room number, collected items, and without domain knowledge are based on compressed grayscale images with discretized pixel intensity into eight levels.

Exploration of the state space is implemented as an iterative process. At each iteration, a cell is sampled from the archive, its state is restored in the environment, and the agent starts exploration with stochastic exploration policy. If the agent visits new cells during the run, they are added to the archive. The statistic of visits is updated for existing cells in each iteration. For both types of representation, the cell stores the highest score that the agent had when it visited the cell. A cell is sampled from the archive by heuristic, preferring more promising cells.

Exploiting domain-specific knowledge makes it difficult to use Go-Explore in a new environment. In our work, we use the idea of clustering of a set of visited states and propose to use a supervised learning model to perform clustering based on the similarity of states. We use a reachability network from the Episodic Curiosity Module [15] as a similarity model predicting similarity score for a pair of states. The clusters are organized into a graph using connectivity information between their states in a similar way as the Memory graph [14] is

built. RND module [4] is used to detect novel states. Our approach doesn't exploit domain knowledge, which allows us to apply RbExplore to the Prince of Persia environment without feature handcrafting.

3   Background

3.1   Markov Decision Processes

A Markov Decision Process (MDP) for a fully observable environment is considered as a model for interaction of an agent with an environment:

$$\mathcal{U} = (\mathcal{S}, \mathcal{A}, p, r, \gamma, s_{init}), \tag{1}$$

$\mathcal{S}$ — a state space, $\mathcal{A}$ — an action space, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ — a state transition distribution, $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ — a reward function, $\gamma \in [0; 1]$ — a discount factor, $s_{init} \in \mathcal{S}$ — an initial state of the environment.

An episode starts in the state s_0. Each step t the agent samples an action a_t based on the current state s_t: $a_t \sim \pi(\cdot|s_t)$, where $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ — a stochastic policy, which defines the conditional distribution over the action space. The environment responds with a reward $r_t = r(s_t, a_t, s_{t+1})$ and moves into a new state $s_{t+1} \sim p(\cdot|s_t, a_t)$. The result of the episode is a return R_0 — a discounted sum of the rewards obtained by the agent during the episode, where $R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i$. Action-value function Q^π is defined as the expected return for using action a_t in a certain state s_t: $Q^\pi(s_t, a_t) = \mathbb{E}_{s_{t+1} \sim p(\cdot|s_t, a_t), a_{t+1} \sim \pi(\cdot|s_{t+1})} [R_t|s_t, a_t]$. State-value function V^π can be defined via action-value function Q^π: $V^\pi(s) = \max_a Q^\pi(s, a)$. The goal of reinforcement learning is to find the optimal policy π^*:

$$\pi^* = \mathrm{argmax}_\pi Q^\pi(s, a) \quad \forall s \in \mathcal{S}, \forall a \in \mathcal{A} \tag{2}$$

3.2   Persistent MDPs

The persistent data structure allows access to any version of it at any time [5]. Inspired by that structures, we propose persistent MDPs for RL. We consider MDP to have persistence property if for any state $s_v \in \mathcal{S}$ exists policy $\pi^p_{s_v}$, which transits agent from the initial state s_{init} to state s_v, in a finite number of timesteps T. Thus, persistent MDP is expressed as:

$$\mathcal{U}^p = (\mathcal{S}, \mathcal{A}, p, r, \gamma, s_{init}, \pi^p), \tag{3}$$

However, the way of returning to visited states can differ. For example, instead of policy $\pi^p_{s_v}$ it could be an environment property, that allows one to save and load states.

4 Exploration via state space clustering

In this paper, we propose the RbExplore algorithm that uses similarity of states to build clustered representation of a set of visited states. There are two essential components of the algorithm: a similarity model, which predicts a similarity measure for a pair of states, and a graph of clusters, which is a clustered representation of a set of visited states organized as a graph. The schema of the algorithm is shown in Figure 1.

A high-level overview of one iteration of the RbExplore algorithm:

1. Generate exploration trajectories: sample M clusters from the graph of clusters $\mathbf{G}$ based on cluster visits statistics (e.g. preferring the least visited clusters), roll back to corresponding states, and run exploration.
2. Generate training data for the similarity model R from the exploration trajectories and additional trajectories starting from novel states filtered by the novelty detection module. Full trajectory prefixes are used to generate negative examples.
3. Train the similarity model R.
4. Update the graph $\mathbf{G}$ with states from the exploration trajectories and merge its clusters. A state is added to the graph $\mathbf{G}$ and forms new clusters if it is dissimilar to states which are already in the graph $\mathbf{G}$. The similarity model is used to select such states.
5. Train the novelty detection module on the states from the exploration trajectories.

As a result of one iteration, novel states are added to the graph $\mathbf{G}$, the statistics of visits to existing clusters are updated, the similarity model and the novelty detection module are trained on the data collected during the current iteration.

4.1 Similarity model

As a feature for clustering, it is proposed to use the distance between states in trajectories. The states located close to each other are considered similar, the states distant from each other are considered dissimilar. A supervised model is used to estimate the similarity measure between states $R : \mathcal{S} \times \mathcal{S} \to [0, 1]$. It takes a pair of states as input and outputs a similarity measure between them. The training dataset is produced by labeling pairs of states for the same trajectory $\{\tau^k = s_1^k, \ldots, s_T^k\}$: triples (s_i^k, s_j^k, y_{ij}^k) are constructed, where y_{ij}^k is a class label. States s_i^k, s_j^k are considered similar $(y_{ij}^k = 1)$ if the distance between them in the trajectory τ^k is less than n steps: $|i - j| < n$. Negative examples $(y_{ij}^k = 0)$ are obtained from pairs of states that are more than N steps apart from each other: $|i - j| > N$. The model R is trained as a binary classifier predicting whether two states are close in the trajectory (class 1) or not (class 0).

Figure 2 illustrates the training data generation procedure. A neural network model is used as a similarity model R as the experiments are performed in environments with high-dimensional state spaces. The network R with parameters
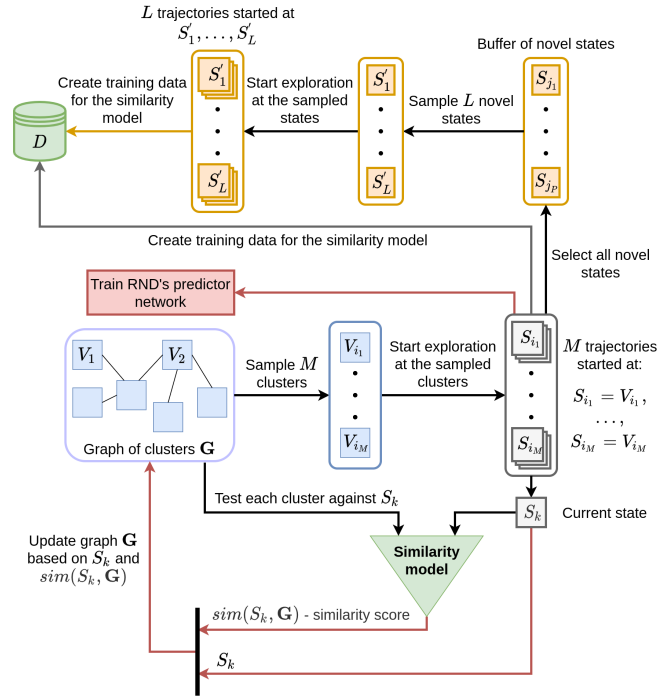
Fig. 1. Scheme of the RbExplore algorithm: M exploration trajectories are generated by running exploration from clusters sampled from **G**. L additional trajectories are generated by running exploration from novel states selected from the exploration trajectories by the novelty detection module. Training data is generated for the similarity model from the exploration trajectories and the additional trajectories. The similarity model is trained on the generated data. **G** is updated based on the states from the exploration trajectories and their similarity score with clusters of **G**. The novelty detection module is trained on the states from the exploration trajectories.

w is trained on the training data set D using binary cross-entropy as a loss function:

$$\mathbb{E}_{(s_1, s_2, y) \sim D} \left[-y \log R_w(s_1, s_2) - (1 - y) \log \left(1 - R_w(s_1, s_2) \right) \right] \to \min_w$$

4.2   Graph of clusters

Clustering of the state space $\mathcal{S}$ is an iterative process using the similarity model R and the chosen stochastic exploration policy $\pi_{explore}(a|s)$ (e.g. uniform distribution over actions). A cluster $v = (s, snap)$ is a pair of state s — the center of the cluster and the corresponding snapshot of the simulator $snap$. Each iteration, a cluster is selected from which exploration will be continued. The state of
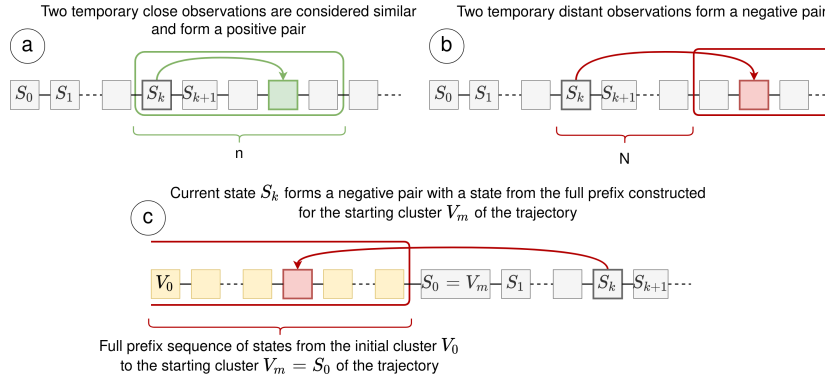
Fig. 2. Generation of training data for the similarity model. **a,b)** Generation of positive and negative examples from states of the same trajectory. **c)** Generation of negative examples using the full prefix of the starting cluster of the trajectory.

the selected cluster is restored in the environment using the corresponding snapshot and the agent starts exploration with stochastic exploration policy $\pi_{explore}$. For each state s_i of the obtained trajectory $\tau = (s_1, snap_1), \ldots, (s_T, snap_T)$ a measure of similarity with the current set of clusters is calculated. A state s_i is considered as belonging to the cluster $v = (s, snap)$ if the measure of similarity between the state and the cluster's state is greater than the selected threshold θ_{sim}: $R(s, s_i) > \theta_{sim}$. Otherwise, a new cluster is created.

Clusters are organized into a directed graph (V, E). Each vertex of the graph corresponds to a cluster. If two successive states s_k and s_{k+1} in the same trajectory $\tau = s_1, \ldots, s_T$ belong to different clusters v_i and v_j, an arc between those clusters (v_i, v_j) is added to the graph. Cluster visit statistics and arc visit statistics are updated each iteration. The graph is initialized with the initial state of the environment $(s_{init}, snap_{init})$. The cluster is selected from the graph for exploration using sampling strategy σ that can take into account the structure of the graph and the collected statistics (e.g. the probability of sampling a cluster is inversely proportional to the number of visits). Each iteration of the graph building procedure can be alternated with training the similarity model R on the obtained trajectories. The search for a cluster to which the current state s_i of the trajectory belongs can be accelerated by considering first those vertices which are adjacent to the cluster to which the previous state s_{i-1} was assigned.

In order to improve the quality of the similarity model R on states from novel regions of the state space $\mathcal{S}$, an RND module is used. For each state, the RND module outputs an intrinsic reward which is used as a measure of the state's novelty. The state is considered novel if the intrinsic reward is greater than $\beta_{intrinsic}$. At each iteration, all states from the trajectory τ, which are detected by the RND module as a novel, are placed into the buffer of novel states B. The buffer B is used to generate additional training data for the similarity model

R which includes states from novel regions of the state space $\mathcal{S}$. A set of states $\{(s_i^*, snap_i^*)\} \sim B$ is randomly sampled from the buffer B, and the agent starts an exploration with an exploration strategy by restoring the sampled simulator states. The resulting trajectories are used solely to generate additional training data for the similarity model R.

When an exploration trajectory is processed and a new cluster v is added to the graph G a new arc from v to the parent cluster from which v was created is added to a set of arcs to parent cluster E_{parent}. A prefix — a sequence of states from the parent cluster to v is stored along with the arc. Thus, for any cluster in the graph G, it is possible to construct a sequence of states that leads to the initial cluster $(s_{init}, snap_{init})$. This property is used to add negative pairs (s_1, s_2) to the training data set of the similarity model R such that $s_1 \in \tau_1$ and $s_2 \in \tau_2$ are distant from each other and $\tau_1 \neq \tau_2$. If the exploration trajectory τ started from a cluster $(s, snap)$, a full prefix $\tau^{prefix} = s_{init}, \ldots, s$ for the trajectory $\tau = (s, snap), \ldots$ is constructed to obtain additional negative examples. For a state $s_1 \in \tau$, a sufficiently distant state $s_2 \in \tau^{prefix}$ is randomly selected to form a negative example (s_1, s_2). Figure 2 (c) illustrates this procedure.

Redundant clusters are created each iteration due to the inaccuracy of the similarity model R. A cluster merge procedure is proposed to mitigate the issue. It tests all pairs of clusters in the graph G and merges the pair $v_1 = (s_1, snap_1), v_2 = (s_2, snap_2)$ into a new cluster if the similarity measure of their states is greater than the selected threshold θ_{merge}: $R(s_1, s_2) > \theta_{merge}$. The new cluster is incident to any arc that was incident to the two original clusters. Cluster visits statistics are summarized during merging. As a state and a snapshot of the new cluster, the states and the snapshot of the cluster that was added to the graph G earlier are selected.

5   The Prince of Persia domain

We evaluate our algorithm on the challenging Prince of Persia game environment, which has over ten complex levels. Figure 3 shows the first level of the game. To pass it, the prince needs to: find a sword, avoiding traps; return to the starting point; defeat the guard, and end level. In most cases, the agent goes to the next level when he passes the final door, which he also needs to open somehow.

The input of the agent is a 96x96 grayscale image. The agent chooses from 7 possible joystick actions no-op, left, right, up, down, A, B. The same action may work differently depending on the situation in the game. For example, the agent can jump forward for a different distance, depending on the take-off run. The agent can jump after using action A, or strike with a sword if in combat. Also, the agent can interact with various objects: ledges, pressing plates, jugs, breakable plates.

The environment is difficult for RL algorithms because it has the action space with changing causal relationships and requires mastery in many game aspects, such as fighting. Also, the first reward is thousands of steps apart from the initial agent position.
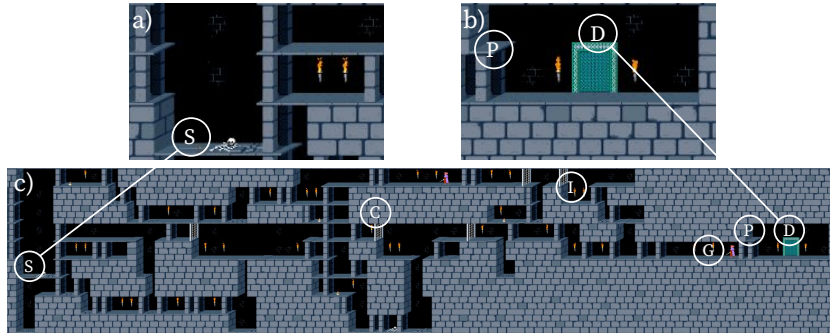
Fig. 3. The Prince of Persia environment. **a,b)** Examples of environment observations from the agent's view. **c)** The complete map of the first level of the game. The agent's task is to get from the initial location (I) to the final door (D). To solve this problem, the agent needs to: pick up a sword (S), go back and defeat a guard (G), stand on the pressure plate (P) to open the door, proceed to the exit. The environment has many obstacles, such as cell doors (C) and various traps.

We use the percentage coverage metric % Cov $= \frac{|U_{visited}|}{|U_{full}|}$, i.e. ratio of coverage to the max coverage in the corresponding level, where $U_{visited}$ is a set of the visited units and U_{full} full coverage set. We consider the minimum unit to be the area, which roughly corresponds to space above the plate. For example, for the first level in the room with the sword, the full area has 36 units, but the agent can visit only 34 of them.

6 Experiments

We evaluate the exploration performance of our RbExplore algorithm on the first three levels of the Prince of Persia environment alongside state-of-the-art curiosity methods ICM and RND.

6.1 Experimental setup

Raw environment frames are preprocessed by applying a frameskip of 4 frames, converting into grayscale, and downsampling to 96x96. Frame pixels are rescaled to the 0-1 range. The neural network R consists of two subnetworks: ResNet-18 network and a four-layer fully connected neural network. ResNet-18 accepts a frame as input and produces its embedding of dimension 512. Embeddings of the pair's frames are concatenated and fed into the fully connected network that performs binary classification of the pair of embeddings.

The same algorithm parameters are used for all levels. The maximum number of frames per exploration trajectory is 1500. Episodes are terminated on the loss of life. Training data for the similarity model R is generated with parameters $n = 5$ and $N = 25$. The similarity threshold $\theta_{similarity} = 0.5$. The similarity

threshold for the cluster merging $\theta_{merge} = 0.5$. Merge is run every 15 iteration. For exploration, clusters are sampled from the graph G with probabilities inversely proportional to the number of visits. Every iteration $M = 30$ clusters are sampled from the graph. The uniform distribution overall actions are used as the exploration policy $\pi_{explore}$. RND module intrinsic reward threshold for detecting novel states $beta_{intrinsic} = 2.5$. $L = 1000$ states are sampled from the buffer of novel states.

To prevent the formation of a large number of clusters at the early stages due to the low quality of the similarity model R, the similarity model is pretrained for 500000 steps with a gradual increase of the value of the similarity threshold from 0 to $\theta_{similarity}$. At the same time, the necessary normalization parameters of the RND module are initialized. After pretraining the graph G is reset to the environment's initial state $(s_{init}, snap_{init})$ and RbExplore is restarted with the fixed similar threshold $\theta_{similairty}$.

6.2   Exploring the Prince of Persia environment

By design of the Prince of Persia environment, the agent's observation does not always contain information about whether the agent carries a sword. To get around this issue the agent starts the first level with the sword. Also, the agent is placed at the point where the sword is located. This location is far enough from the final door so reaching the final door is still a challenging task. For the other levels, we did not make any changes to the initial state.

Evaluation of RbExplore, ICM, and RND on the first three levels of the Prince of Persia environment is shown in Figure 5. On the first level, RbExplore performed significantly outperforms ICM and RND, and also have visited all possible rooms of the level. The visualization of the coverage is shown in Figure 4.

In levels 2 and 3, none of the algorithms were able to visit all the rooms in 15 million steps. On level two RbExpore shows slightly worse results than RND and ICM. We explain it by the fact that the learning process in RND and ICM is driven by an exploration bonus, which helps them to explore local areas inside rooms more accurately. Each of the algorithms was able to visit only 7 rooms at the very beginning of the level. On level 3 RbExplore shows slightly better coverage than RND and both of them outperform ICM.
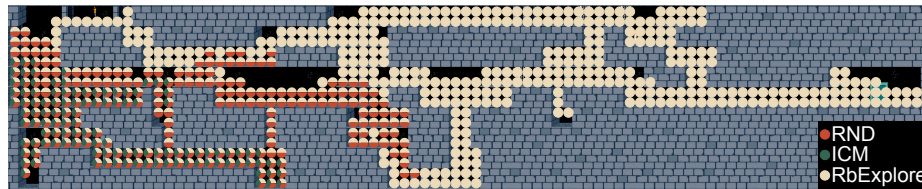

Fig. 4. The visualization of the first level coverage for the best RND, ICM, and RbExplore runs. RbExplore significantly outperforms RND and ICM.
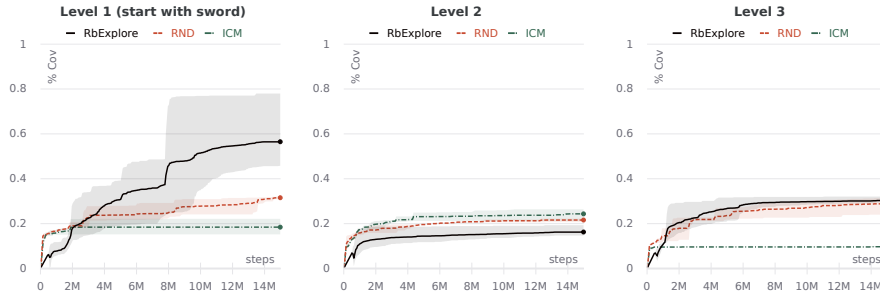
Fig. 5. Performance of RbExplore, RND, and ICM for the first three levels of the Prince of Persia environment. **Left:** Level 1 - RbExplore significantly outperforms RND and ICM; **Center:** Level 2 - All methods resulted in visits to 7 rooms. RND and ICM outperform RbExplore as they deal better with local explorations and explore rooms more thoroughly; **Right:** Level 3 - RbExplore significantly outperforms ICM and show comparable performance with RND; Curves are averaged over three runs. The shading indicates the min-max range.

6.3   Ablation study

In order to evaluate the contribution of each component of the RbExplore algorithm, we perform an ablation study. Experiments were run on level one with two versions of RbExplore. The first version doesn't merge clusters in the graph G. The second one doesn't build the full trajectory prefix when generating negative examples for the training data of the similarity model, thus, states of negative pairs are sampled only from the same trajectory. Figure 6 shows that disabling these components hurts the performance of the algorithm.

Additional experiments were run on the level one to study the impact of the value of θ_{merge} parameter on the performance. Figure 6 shows that the performance of RbExplore with $\theta_{merge} \in \{0.25, 0.75\}$ is worse than that of RbExplore with $\theta_{merge} = 0.5$.

7   Conclusion

In this paper, we introduce a pure exploration algorithm RbExplore that uses the formalized version of resettable environment named persistent MDP. The experiments showed that RbExplore coupled with a simple exploration policy, which is a uniform distribution over actions, demonstrates performance comparable with RND and ICM methods in the hard-exploration environment Prince of Persia in a no-reward setting. RbExplore, ICM, and RND got stuck on the second and third levels roughly in the same locations where the agent must perform a very specific sequence of actions over a long-time horizon to go further. The combining of RbExplore exploration and exploitation of RL approaches, which also utilize pMDPs, is an important direction for future work to resolve this problem.
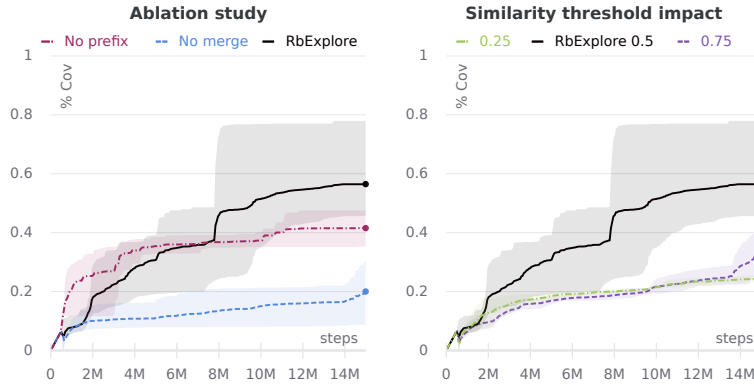
Fig. 6. Left: Level 1 - Comparison of RbExplore with its versions which don't merge clusters (No merge) and don't build the full trajectory prefix to generate negative example for the training data for the similarity model (No prefix). Disabling merge procedure and generation of negative examples with the use of trajectory prefixes hurts performance of the RbExplore algorithm; **Right:** Level 1 - Comparison of RbExplore with similarity thresholds for merging $\theta_{merge} \in \{0.25, 0.5, 0.75\}$. The performance of RbExplore with $\theta_{merge} \in \{0.25, 0.75\}$ is worse than that of RbExplore with $\theta_{merge} = 0.5$; Curves are averaged over three runs. The shading indicates the min-max range.

References

1. Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., Mc-Grew, B., Tobin, J., Pieter Abbeel, O., Zaremba, W.: Hindsight experience replay. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30, pp. 5048–5058. Curran Associates, Inc. (2017), `https://proceedings.neurips.cc/paper/2017/file/453fadbd8a1a3af50a9df4df899537b5-Paper.pdf`
2. Bellemare, M.G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., Munos, R.: Unifying count-based exploration and intrinsic motivation. In: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. pp. 1471–1479 (2016), `https://proceedings.neurips.cc/paper/2016/hash/afda332245e2af431fb7b672a68b659d-Abstract.html`
3. Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., Efros, A.A.: Large-scale study of curiosity-driven learning. In: ICLR (2019)
4. Burda, Y., Edwards, H., Storkey, A., Klimov, O.: Exploration by random network distillation. In: International Conference on Learning Representations (2019), `https://openreview.net/forum?id=H1lJJnR5Ym`
5. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making data structures persistent. Journal of computer and system sciences **38**(1), 86–124 (1989)

6. Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K.O., Clune, J.: Go-explore: a new approach for hard-exploration problems (2021)
7. Fang, M., Zhou, T., Du, Y., Han, L., Zhang, Z.: Curriculum-guided hindsight experience replay. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32, pp. 12623–12634. Curran Associates, Inc. (2019), `https://proceedings.neurips.cc/paper/2019/file/83715fd4755b33f9c3958e1a9ee221e1-Paper.pdf`
8. Florensa, C., Held, D., Geng, X., Abbeel, P.: Automatic goal generation for reinforcement learning agents. In: Dy, J., Krause, A. (eds.) Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 80, pp. 1515–1528. PMLR, Stockholmsmässan, Stockholm Sweden (7 2018), `http://proceedings.mlr.press/v80/florensa18a.html`
9. OpenAI, Andrychowicz, M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., Schneider, J., Sidor, S., Tobin, J., Welinder, P., Weng, L., Zaremba, W.: Learning dexterous in-hand manipulation (2019)
10. Oudeyer, P.Y., Kaplan, F.: How can we define intrinsic motivation ? In: Proceedings of the 8th International Conference on Epigenetic Robotics: Modeling CognitiveDevelopment in Robotic Systems (2008)
11. Pathak, D., Agrawal, P., Efros, A.A., Darrell, T.: Curiosity-driven exploration by self-supervised prediction. In: ICML (2017)
12. Racaniere, S., Lampinen, A., Santoro, A., Reichert, D., Firoiu, V., Lillicrap, T.: Automated curriculum generation through setter-solver interactions. In: International Conference on Learning Representations (2020), `https://openreview.net/forum?id=H1e0Wp4KvH`
13. Ren, Z., Dong, K., Zhou, Y., Liu, Q., Peng, J.: Exploration via hindsight goal generation. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 32, pp. 13485–13496. Curran Associates, Inc. (2019), `https://proceedings.neurips.cc/paper/2019/file/57db7d68d5335b52d5153a4e01adaa6b-Paper.pdf`
14. Savinov, N., Dosovitskiy, A., Koltun, V.: Semi-parametric topological memory for navigation. In: International Conference on Learning Representations (2018), `https://openreview.net/forum?id=SygwwGbRW`
15. Savinov, N., Raichuk, A., Vincent, D., Marinier, R., Pollefeys, M., Lillicrap, T., Gelly, S.: Episodic curiosity through reachability. In: International Conference on Learning Representations (2019), `https://openreview.net/forum?id=SkeK3s0qKQ`
16. Skrynnik, A., Panov, A.I.: Hierarchical Reinforcement Learning with Clustering Abstract Machines. In: Kuznetsov, S.O., Panov, A.I. (eds.) Artificial Intelligence. RCAI 2019. Communications in Computer and Information Science. vol. 1093, pp. 30–43. Springer (2019). https://doi.org/10.1007/978-3-030-30763-9_3
17. Xu, K., Verma, S., Finn, C., Levine, S.: Continual learning of control primitives: Skill discovery via reset-games (2020)
18. Zhu, H., Yu, J., Gupta, A., Shah, D., Hartikainen, K., Singh, A., Kumar, V., Levine, S.: The ingredients of real world robotic reinforcement learning. In: International Conference on Learning Representations (2020), `https://openreview.net/forum?id=rJe2syrtvS`