Rank the Episodes: A Simple Approach for Exploration in Procedurally-Generated Environments

Daochen Zha, Wenye Ma, Lei Yuan, Xia Hu, Ji Liu

Department of Computer Science and Engineering, Texas A&M University
AI Platform, Kwai Inc.

Email: daochen.zha@tamu.edu

Code: https://github.com/daochenzha/rapid
**Background: Exploration is an Open Challenge for RL**

- **What is exploration?** The ability of the agents to discover novel states in the environments.

- **Why we need exploration?** Learning a good policy with reinforcement learning relies on the ability of discovering the novel states in the first place (the rewards can be sparse).

- **How we can encourage exploration?** The most popular method for encouraging exploration is to give intrinsic rewards based on uncertainty, e.g., count-based exploration [1], curiosity driven exploration [2], etc.

---

Motivation: Exploration Needs to Generalize

- **Overfitting Issue:** While Deep RL has made impressive progress in recent years, most of the previous studies use the same singleton environment for training and testing. Recent studies show that the agent trained in this way is susceptible to overfitting and may fail to generalize to even a slightly different environment [1][2].

- **Procedurally-Generated Environments:** To address this issue, some procedurally-generated environments are proposed, where a different environment is generated in each episode.

Motivation: Can Intrinsic Rewards Generalize?

• **1D Maze as a Motivating Examples:** The agent (red) needs to move right in each step to reach a goal.

• **Count-Based Exploration in Procedurally-Generated Environments:** It is less effective because visiting a novel state does not necessarily mean a good exploration behavior.
Motivation: Episode-Level Exploration

- **Why Humans can Generalize?** Because humans often view the agent in episode-level rather than state-level. For example, one can easily tell whether an agent has well explored an environment by looking into the coverage rate of the current episode, even if the current environment is totally different from the previous ones.

![Sample episodes diagram]

- **Visited**
- **Not visited**

Sample episodes:
- Episodic score = 0.25 Poor exploration
- Episodic score = 0.875 Good exploration
A New Exploration Strategy that Can Generalize

• **Episode-Level Exploration:** *We propose Exploration via Ranking the Episodes (RAPID)*

Figure 2: An overview of RAPID. The past episodes are assigned episodic exploration scores based on the local view, the global view, and the extrinsic reward. Those highly scored episodes are stored in a small ranking buffer. The agent is then encouraged to reproduce the past good exploration behaviors, i.e., the episodes in the buffer, with imitation learning.
Episodic Exploration Score

- **Local Score:** A local view of an episode. The number of district states in an episode divided by the total number of states in this episode.

  \[ S_{\text{local}} = \frac{N_{\text{distinct}}}{N_{\text{total}}} \]

- **Global Score:** A score that also considers previous episodes. The mean count-based score of the current episode.

  \[ S_{\text{global}} = \frac{1}{N_{\text{total}}} \sum_{s} \frac{1}{\sqrt{N(s)}} \]

- **Episodic Exploration Score:** We further consider the extrinsic rewards. The final score is the weighted sum of the three scores.

  \[ S = w_0 S_{\text{ext}} + w_1 S_{\text{local}} + w_2 S_{\text{global}} \]
Algorithm 1 Exploration via Ranking the Episodes (RAPID)

1: Input: Training steps $S$, buffer size $D$, RL rollout steps $T$
2: Initialize the policy $\pi_\theta$, replay buffer $D$
3: for iteration = 1, 2, ... until convergence do
4:   Execute $\pi_\theta$ for $T$ timesteps
5:   Update $\pi_\theta$ with RL objective (also update value functions if any)
6:   for each generated episode $\tau$ do
7:     Compute episodic exploration score $S_\tau$ based on Eq. (4)
8:     Give score $S_\tau$ to all the state-action pairs in $\tau$ and store them to the buffer
9:     Rank the state-action pairs in $D$ based on their exploration scores
10:    if $D.length > D$ then
11:       Discard the state-action pairs with low scores so that $D.length = D$
12:    end if
13:    for step = 1, 2, ..., $S$ do
14:       Sample a batch from $D$ and train $\pi_\theta$ using the data in $D$ with behavior cloning
15:    end for
16: end for
17: end for
Results on MiniGrid Benchmarks

(a) MultiRoom-N7-S4  (b) MultiRoom-N10-S4  (c) MultiRoom-N7-S8  (d) MultiRoom-N10-S10

(e) KeyCorridor-S3-R2  (f) KeyCorridor-S3-R3  (g) KeyCorridor-S4-R3  (h) MultiRoom-N12-S10
Results on MiniGrid Benchmarks

- **Observation 1:** **RAPID** is fast, accelerating the convergence with up to **10X**.
- **Observation 2:** **RAPID** is effective. Only one that works in **KeyCorridor-S4-R3**.
Observation: RAPID also works with raw images as inputs.
Results on Sparse MuJoCo Tasks

- **Setting**: The rewards are delayed until the last timestep of an episode.
- **Observation**: RAPID also works with continuous action spaces.
Ablation Study

- **Observation 1:** All the three scores and the ranking buffer helps.
- **Observation 2:** Local score plays a key role (most of previous work only considers global view).

<table>
<thead>
<tr>
<th></th>
<th>RAPID</th>
<th>w/o local</th>
<th>w/o global</th>
<th>w/o reward</th>
<th>w/o buffer</th>
<th>w/o ranking</th>
</tr>
</thead>
<tbody>
<tr>
<td>MR-N7S4</td>
<td>0.787 ± 0.001</td>
<td>0.787 ± 0.000</td>
<td>0.787 ± 0.001</td>
<td>0.781 ± 0.002</td>
<td>0.000 ± 0.000</td>
<td>0.002 ± 0.002</td>
</tr>
<tr>
<td>MR-N10S4</td>
<td>0.778 ± 0.000</td>
<td>0.778 ± 0.000</td>
<td>0.778 ± 0.001</td>
<td>0.775 ± 0.002</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td>MR-N7S8</td>
<td>0.678 ± 0.001</td>
<td>0.677 ± 0.002</td>
<td>0.677 ± 0.002</td>
<td>0.652 ± 0.004</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td>MR-N10S10</td>
<td>0.632 ± 0.001</td>
<td>0.238 ± 0.288</td>
<td>0.630 ± 0.002</td>
<td>0.604 ± 0.010</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td>MR-N12S10</td>
<td>0.644 ± 0.001</td>
<td>0.001 ± 0.001</td>
<td>0.633 ± 0.005</td>
<td>0.613 ± 0.007</td>
<td>0.000 ± 0.000</td>
<td>0.000 ± 0.000</td>
</tr>
<tr>
<td>KC-S3R2</td>
<td>0.934 ± 0.004</td>
<td>0.933 ± 0.002</td>
<td>0.934 ± 0.000</td>
<td>0.929 ± 0.003</td>
<td>0.018 ± 0.008</td>
<td>0.527 ± 0.380</td>
</tr>
<tr>
<td>KC-S3R3</td>
<td>0.922 ± 0.001</td>
<td>0.885 ± 0.022</td>
<td>0.912 ± 0.003</td>
<td>0.903 ± 0.002</td>
<td>0.012 ± 0.007</td>
<td>0.013 ± 0.006</td>
</tr>
<tr>
<td>KC-S4R3</td>
<td>0.473 ± 0.087</td>
<td>0.150 ± 0.095</td>
<td>0.244 ± 0.144</td>
<td>0.035 ± 0.035</td>
<td>0.000 ± 0.000</td>
<td>0.001 ± 0.001</td>
</tr>
</tbody>
</table>

Table 1: Performance of RAPID and the ablations on MiniGrid environments. The mean maximum returns and standard deviations are reported. MR stands for MultiRoom; KC stands for KeyCorridor. Learning curves are in Appendix C.
Observation 1: RAPID works well even without extrinsic rewards.

Observation 2: RIDE is indirectly optimizing local exploration score.
Some insights:

1. Episode-level score could better generalize in procedurally-generated environments.
2. Episode-level exploration also works well in singleton setting (e.g., MuJoCo) and continuous state/action spaces.
3. Ranking and imitating good exploration behaviors is very effective (a good exploration behavior could be leveraged more than once)

Our contributions:

1. We propose a practical algorithm, named RAPID, for efficient exploration.
2. Unlike traditional state-level intrinsic rewards, we approach sample efficiency for hard-exploration problems in episode-level.
3. Our code is made publicly available.

Code: https://github.com/daochenzha/rapid