On the Complexity of Exploration in Goal-Driven Navigation

Maruan Al-Shedivat[1*] **Lisa Lee**[1*] **Ruslan Salakhutdinov**[1,2] **Eric P. Xing**[1,3]
[1]Carnegie Mellon University, [2]Apple, [3]Petuum Inc.

Abstract

Building agents that can explore their environments intelligently is a challenging open problem. In this paper, we make a step towards understanding how a hierarchical design of the agent's policy can affect its exploration capabilities. First, we design `EscapeRoom` environments, where the agent must figure out how to navigate to the exit by accomplishing a number of intermediate tasks (*subgoals*), such as finding keys or opening doors. Our environments are procedurally generated and vary in complexity, which can be controlled by the number of subgoals and relationships between them. Next, we propose to measure the complexity of each environment by constructing dependency graphs between the goals and analytically computing *hitting times* of a random walk in the graph. We empirically evaluate Proximal Policy Optimization (PPO) with sparse and shaped rewards, a variation of policy sketches, and a hierarchical version of PPO (called HiPPO) akin to h-DQN. We show that analytically estimated *hitting time* in goal dependency graphs is an informative metric of the environment complexity. We conjecture that the result should hold for environments other than navigation. Finally, we show that solving environments beyond certain level of complexity requires hierarchical approaches.

1 Introduction

Deep reinforcement learning research has led us to discover general-purpose algorithms for learning how to control robots [1] and solve games [2, 3], surpassing human abilities. These results indicate a significant progress in the field. However, building agents capable of intelligent exploration even in simple environments is still an unreached milestone.

To make progress towards this goal, first we need to understand and be able to measure when exploration is necessary. For instance, while Atari games seem like a challenging benchmark, it turns out that having a memoryless reactive policy is often sufficient for solving most of these games [4]. On the other hand, there are environments (*e.g.*, Montezuma's Revenge) that can only be solved by achieving some intermediate goals (*subgoals*). Learning about the dependencies between the subgoals requires executing a consistent exploration strategy, reasoning, and multi-step planning, beyond vanilla deep RL methods.

Broadly, exploration is a mechanism used by an agent to reduce uncertainty about its environment (*i.e.*, rewards and state transitions). Notable approaches to exploration include: (1) *count-based* and *intrinsic motivation* methods [5], where the agent (approximately) quantifies uncertainty of the states and actions and tends to visit the states it is least certain about; and (2) various policy-perturbation heuristics, such as ε-greedy, Boltzmann, and parameter-noise methods [6, 7]. All these approaches function on the level of atomic actions and hence are limited when it comes to complex structured tasks with delayed and sparse rewards. To overcome such limitations, it is possible to use the framework of temporal abstractions (*options*) [8, 9]. In particular, Kulkarni et al. [10] argued for hierarchical methods that enable exploration in the *space of goals*, which is also our focus.

*Equal contribution. Correspondence: {alshedivat,lslee}@cs.cmu.edu.

Relational Representation Learning Workshop (NIPS 2018), Montréal, Canada.

In this paper, we aim to understand and measure the complexity of exploration in environments with multiple dependent subgoals, and the effects of hierarchical design of the agent's policy in such environments. To do so, we introduce a collection of procedurally generated, simple grid-world environments called `EscapeRooms` (Table 1). We represent the goal space with dependency graphs, and propose to measure complexity of exploration as the time it takes a random walk in this abstract space to reach the final goal state from the start state in expectation (*i.e.*, the *hitting time*). This measure captures a simple intuition: the more complex the goal dependencies are, the more time it would take the agent to explore how to solve the environment.

To verify that the hitting time is a useful measure of complexity in RL scenarios, we train a few hierarchical and non-hierarchical policies using methods based on proximal policy optimization [PPO, 11] on `EscapeRooms` and measure their exploration capabilities. We use metrics such as success rate and the number of timesteps it takes the agent to achieve each goal. Our results demonstrate that information about the goals is crucial to enable learning in our environments. Moreover, we show that our complexity measure correlates with the performance of the policies—agents perform worse and hierarchy becomes more important in environments with higher exploration complexity.

2 Methods

Given an environment, we would like to quantify *how much exploration is needed to solve the task*. In this section, we introduce the notion of goal-dependency graphs, describe `EscapeRoom` environments, and compute different measures of the exploration complexity for these environments.

2.1 Goal-dependency graphs & exploration complexity

We are interested in a scenario where the agent can achieve the final goal only after having accomplished a number of intermediate goals. Assuming that the goals and dependencies are given, we can construct a graph $G(V, E)$ with nodes V representing the goals and edges E representing the relationships between the goals. From this perspective, we can treat the agent executing a (stochastic) policy in an environment with these subgoals as a random walk on the corresponding goal-dependency graph. To measure complexity of exploration in the given environment, we introduce the following notation. Let n be the number of nodes in the graph, and $W \in \mathbb{R}^{n \times n}$ the adjacency matrix of the graph weighted by the probabilities of transition from one goal to the other (according to the policy π executed by the agent). Let $D \in \mathbb{R}^{n \times n}$ be the corresponding diagonal weighted degree matrix:

$$D_{ii} := \sum_{j=1}^{n} W_{ij}, \quad D_{ij} := 0, \, \forall i \neq j$$

Now, we can use the graph Laplacian, $L := W - D$, to compute the expected time it would take the random walk to reach a given goal node with index t from the initial node with index s in the graph for the first time (also known as the *hitting time*) [12]. To do so, we can solve the following linear system (subscripts denote indices):

$$\begin{aligned} Lx &= b \quad \text{s.t. } x_t = 0, \\ \text{where} \quad b_s &= 1, \, b_t = -1, \, b_k = 0 \quad \forall k \notin \{s, t\} \end{aligned} \tag{1}$$

where $x, b \in \mathbb{R}^n$. The solution, $x^\star_s$, will be the hitting time from s to t. Solving (1) for each goal in the graph allows us to analytically compute different statistics for any goal-dependency graph under a given random walk, *e.g.*, the expected number of states reachable under a given time limit.

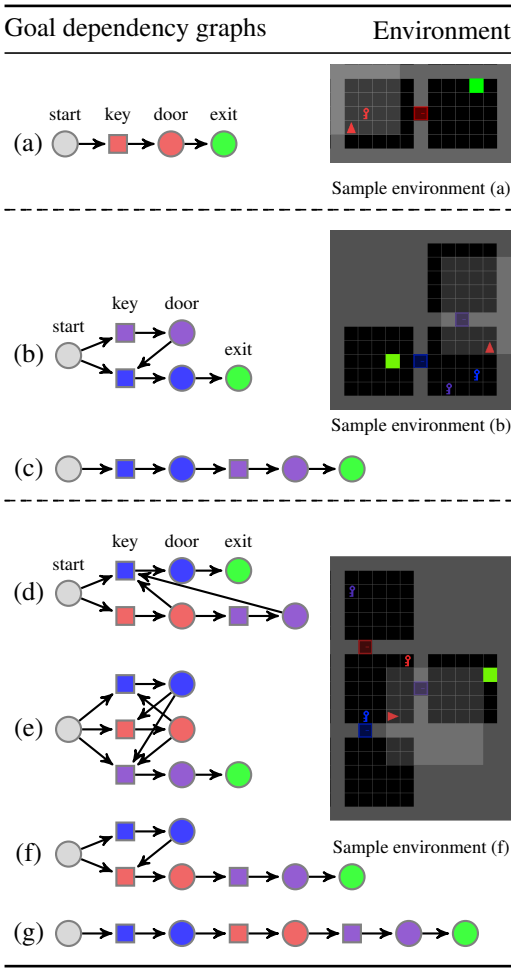
2.2 `EscapeRoom` environments

We design a set of grid-world environments (Table 1) where the agent must pick up keys and open locked doors (*i.e.*, accomplish intermediate goals) in order to reach the exit (the final goal). The agent has 5 actions: `move-forward`, `turn-left`, `turn-right`, `pick-up` (key), and `open` (door). The `pick-up` action only succeeds if the key is in front of the agent. The `open` action only succeeds if a locked door is in front of the agent and the agent has already picked up a key of the same color as the door. Upon arriving at the exit, the agent receives a reward of 1 and the episode terminates. Our `EscapeRoom` environments are based on Gym MiniGrid [13] and follow the OpenAI Gym API [14].

In each episode, we procedurally generate a new environment; the object locations, colors of the keys and doors, and the room layouts are all randomized. The agent always begins at a random cell in the center room, which branches out to 1-3 other rooms, one of which contains the exit. Each of the branching rooms is initially blocked by a locked door, so an environment with n rooms has exactly $n-1$ keys and doors. Each open cell can contain up to one of 3 object types (`Exit`, `Key`, or `Door`) with one of 6 possible colors. The environment is partially observable, meaning that the agent can only observe its local surroundings and cannot see through walls. In our experiments, each observation is a $7 \times 7 \times 3$ array representing the 7×7 view in front of the agent with three channels (object IDs, color IDs, and a binary matrix capturing whether a door is open).

Goal dependencies in `EscapeRooms`. In Table 1, we enumerate all possible goal dependency graphs for different `EscapeRoom` environments with up to 4 rooms. Each goal is represented as one-hot encodings of (color, object); for example, (yellow key) means to pick up the yellow key, and (blue door) means to open the blue door. To understand how complex each environment is from the stand point of exploration in the goal space, we compute the hitting time (HT) for each graph (Table 2). Note that dependency graphs in Table 1 are simplified for illustration purposes (each goal node is assumed to be visited only once). Computing properties of random walks requires strongly connected graphs, and hence we construct augmented goal-dependency graphs and use those for estimating the hitting times of interest (see Appendix A).

Table 1: `EscapeRoom` environments. On the right, we enumerate all possible dependency graphs (up to a permutation of colors) for environments with two rooms (a), three rooms (b, c), and four rooms (d, e, f ,g). Each node in the dependency graph can be traversed at most once (*i.e.*, no cyclic paths are allowed). The agent (red triangle) must pick up keys and open locked doors in order to reach the exit (green square). Each door can only be opened by a key of the corresponding color.



Complexity of `EscapeRooms`. Based on Table 2, we make a few observations. First, longer paths from the start to the exit nodes result in slower discovery of how to solve the environment. Similarly, adding alternative paths that do not lead to the exit (proportional to the graph width) also increase the complexity and is reflected by the hitting time metric. Finally, the environment complexity depends not only on the spatial map design but also on the action space. We experimented with adding an extra `drop` (key) action which significantly increased the hitting time for the exit node in goal-dependency graph (Table 2, last row).

2.3 RL algorithms

In this work, we focus on a class of policy gradient methods known as Proximal Policy Optimization (PPO) algorithms [11]. First, we evaluate the vanilla PPO trained using two different reward functions: (1) **PPO** is trained using sparse rewards, where the agent receives +1 reward upon achieving the final goal (*e.g.*, reaching the exit); and (2) **PPO+Bonus** is trained using reward shaping where in addition to the reward for achieving the final goal, the agent also receives +1 reward for achieving intermediate goals (*e.g.*, picking up keys or opening doors).

Next, we introduce a variant of PPO called **HiPPO (Hierarchical PPO)** which borrows the hierarchical framework from [10], but replaces the hierarchical value functions in the h-DQN with hierarchical PPO policies. In more detail, HiPPO uses a *meta-controller* policy to choose intermediate goals for the lower-level *controller* policy to achieve[2]. The controller receives one-hot encoded goals as part of its observation and *intrinsic* rewards for achieving intermediate goals chosen by the meta-controller. The meta-controller receives sparse *extrinsic* rewards from the environment for achieving the final goal and is prompted to submit a new action (*i.e.*, a new goal) each time the lower-level controller accomplishes the previous goal. The pseudocode for HiPPO is given in Algorithm 1. In our experiments, we used a fixed meta-controller that chooses a sequence of goals along a random depth-first search path on the goal dependency graph, rather than a trainable meta-controller policy.

Algorithm 1 Hierarchical PPO

Input: Meta-controller policy π_M, Controller policy π_C
for $i = 1$ **to** num_episodes **do**
  subgoal $g \leftarrow \pi_M(s)$
  while s is not terminal **do**
    $F \leftarrow 0$
    $s_0 \leftarrow s$
    while not (g is reached) **do**
      $a \leftarrow \pi_C(\{s, g\})$
      state s', reward $f \leftarrow \text{Env}(a)$
      intrinsic reward $r \leftarrow \text{Critic}(s', g)$
      PPO_update(π_C, s, a, s', r)
      $F \leftarrow F + f$
      $s \leftarrow s'$
    end while
    PPO_update(π_M, s_0, g, s', F)
    subgoal $g \leftarrow \pi_M(s)$
  end while
end for

Lastly, **PPO+Sketch** is a variation of policy sketches [15] where the agent is provided with a sequence of goals that leads to achieving the final goal. PPO+Sketch is identical to PPO except that in each timestep, the current observation is concatenated with the current intermediate goal[3], *i.e.*, the actions produced by the policy are always conditional on the current goal. Similar to PPO, and unlike HiPPO and PPO+Bonus, PPO+Sketch does not use intrinsic rewards for achieving intermediate goals.

3 Experiments

We evaluate PPO, PPO+Bonus, PPO+Sketch, and HiPPO on `EscapeRoom` environments (a)-(g). We limit the episode length to 1000 time steps. For each method and environment, we use the LSTM policy with hidden dimension 64, and train for 10M total time steps on 128 vectorized environments using the Adam optimizer, learning rate 2.5e-4, discount factor $\gamma = 0.9$, and TD $\lambda = 0.95$. We evaluated each method and environment over 5 trials with different random seeds.

In Figure 1, we see that HiPPO consistently achieves the smallest average episode length and highest success rate on all environments, thus demonstrating the benefit of using hierarchical policies that operate at different temporal scales. Surprisingly, PPO with sparse rewards performs better than PPO+Bonus, showing that the bonus rewards for achieving intermediate goals does not help a non-hierarchical policy. We also find that PPO+Sketch performs worse than PPO indicating that merely conditioning on subgoals might be suboptimal and destructively interferes with optimization.

Table 2: Depth, width, and hitting time (HT) statistics computed for `EscapeRoom` environments (a)-(g).

	(a)	(b)	(c)	(d)	(e)	(f)	(g)
exit depth	2	2	4	2	2	4	6
graph width	1	2	1	2	3	2	1
HT (w/o drop-key)	8.4	12.1	15.1	13.1	13.9	29.2	27.5
HT (w/ drop-key)	16.5	25.2	39.5	27.5	26.7	86.1	82.5

Environments (f) and (g) are more challenging for RL agents due to greater exit depth of their goal dependency graphs, *i.e.*, the agent must achieve a longer sequence of intermediate goals before it can reach the exit. Similarly, the width of the dependency graph introduces complexity (due to paths that don't lead anywhere), but not as much as the depth. We find that the analytically estimated hitting times given in Table 2 are in agreement with the observed empirical performance of the RL algorithms. We also note that despite the complexity of the environments, HiPPO is still able to make some progress on (f) and (g), while the other flat PPO baselines (with or without reward shaping and/or policy sketches) fail to solve them (Figure 1).

[2]The action space of the meta-controller is the space of goals. The controller uses available primitive actions.

[3]Feeding goals as observations into the policy network is slightly different from the original design of Andreas et al. [15]. We plan to investigate the original policy sketch architecture in future work.

Table 3: **Left**: Average success rate (%) to reach the final goal over the last 10 training episodes. **Right**: Average episode length (% of the max length, smaller is more efficient) over the last 10 training episodes. "–" indicates that the method failed to reach the final goal within 1000 steps.

	Average Success Rate							Average Episode Length						
	(a)	(b)	(c)	(d)	(e)	(f)	(g)	(a)	(b)	(c)	(d)	(e)	(f)	(g)
PPO	56.1	28.2	0.2	22.7	19.1	0.0	0.0	78.5	88.0	–	90.7	91.4	–	–
PPO+Bonus	9.0	6.0	0.0	11.0	4.5	0.5	0.0	97.8	96.7	99.9	97.3	98.0	99.9	–
PPO+Sketch	23.2	14.5	0.4	12.6	10.7	0.1	0.0	91.9	94.4	99.9	95.0	95.7	–	–
HiPPO	**74.9**	**57.0**	**60.8**	**48.0**	**29.9**	**11.2**	**19.0**	**48.2**	**67.4**	**69.1**	**71.0**	**85.3**	**96.4**	**93.8**

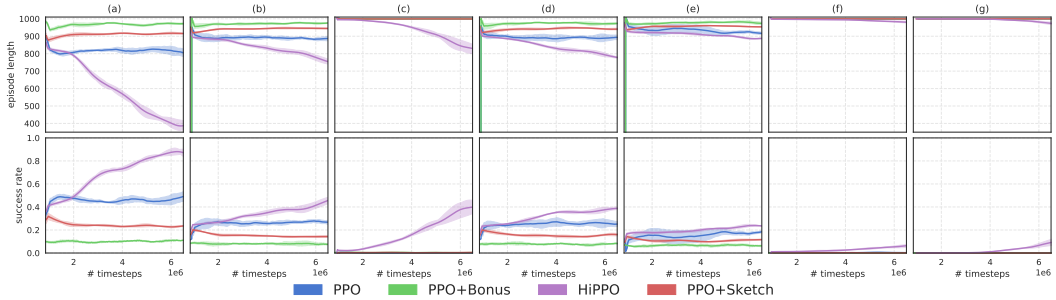


Figure 1: Average episode length and success rate on `EscapeRoom` environments with goal dependency graphs (a)-(g) from Table 1. In all environments, HiPPO achieves the best performance (smallest episode length and highest success rate). In the most complex environments (f) and (g), HiPPO still makes some learning progress.

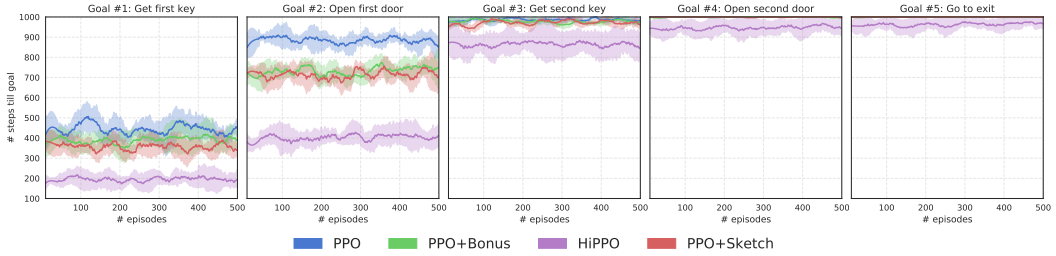


Figure 2: Average number of timesteps to reach each intermediate goal on `EscapeRoom` (c). HiPPO is the quickest method to achieve each goal.

In Figure 3, we illustrate the correlation between the hitting time on goal dependency graphs (Table 2) and the empirical performance of HiPPO (Table 3) for different `EscapeRoom` environments, which demonstrates that analytically estimated hitting time is an informative metric for measuring the complexity of an environment.

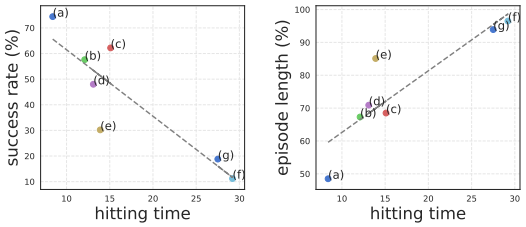


Figure 3: Correlation between the hitting time (Table 2) vs. the average success rate and average episode length (Table 3) of HiPPO for `EscapeRoom` environments (a)-(g) from Table 1. This verifies that the hitting time is a useful measure of complexity for RL environments.

4 Discussion

We designed a simple grid-world `EscapeRoom` environment where it is easy to measure the exploration complexity by analyzing the corresponding goal dependency graphs. We showed that hitting times in goal dependency graphs are consistent with the empirical performance of PPO-based methods, and is therefore a useful metric to measure the complexity of the environment. Finally, we showed the performance improvement of HiPPO over other flat PPO baselines, demonstrating the benefit of using hierarchical policies that operate at different temporal scales.

References

[1] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[4] Christoph Dann, Katja Hofmann, and Sebastian Nowozin. Memory lens: How much memory does an agent use? *arXiv preprint arXiv:1611.06928*, 2016.

[5] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479, 2016.

[6] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

[7] Matthias Plappert, Rein Houthooft, Prafulla Dhariwal, Szymon Sidor, Richard Y Chen, Xi Chen, Tamim Asfour, Pieter Abbeel, and Marcin Andrychowicz. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

[8] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

[9] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, pages 1726–1734, 2017.

[10] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in neural information processing systems*, pages 3675–3683, 2016.

[11] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[12] László Lovász. Random walks on graphs. *Combinatorics, Paul erdos is eighty*, 2(1-46):4, 1993.

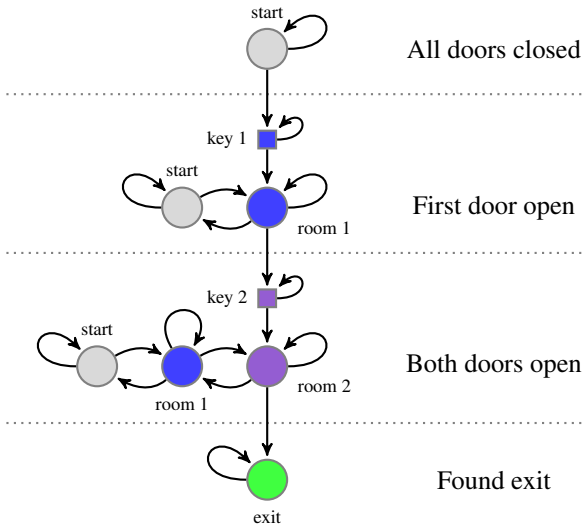
[13] Maxime Chevalier-Boisvert and Lucas Willems. Minimalistic Gridworld Environment for OpenAI Gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[14] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[15] Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. *arXiv preprint arXiv:1611.01796*, 2016.

A Details on computing hitting times

As we mentioned in Section 2.3, to compute the hitting time of random walk we need an augmented goal-dependency graph (which can be generated procedurally from the graphs given in the main text). An example augmented graph for `EscapeRoom` (c) from Table 1 is presented below.



The main difference from the goal dependency graph given in Table 1 is that when the agent picks up a key and opens the corresponding door, it transitions into a subgraph that corresponds to the new layout of the rooms accessible to the agent. Self-loops and transitions between the rooms represent the moving behavior.

We set the following parameters for the random walk. With 80% chance, no transition happens. With 19% chance, the walk transitions from the current node along one of the outgoing edges. Finally, to ensure strong connectivity of the graph, we add 1% chance of the agent moving back to the root start node from any other node in the graph[4]. This corresponds to the situation where the agent is not able to reach the exit within the time limit and must start a new episode.

[4] A similar approach is taken by the PageRank algorithm.