1. Go-Explore: a New Approach for Hard-Exploration Problems

2. First return, then explore

Article | Published: 24 February 2021

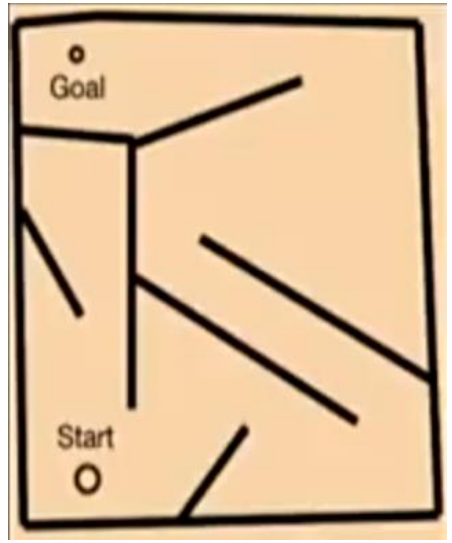
First return, then explore

Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O. Stanley & Jeff Clune

Grand Challenge in RL: Effective Exploration

Hard-Exploration Problems

- Sparse-reword problems
  - rare feedback
- Deceptive problems
  - wrong feedback
  - local optima



reward = distance to goal

background

Typical approach: intrinsic motivation (curiosity、novelty-seeking)

Problems of IM:

1. Detachment : an agent driven by IM could become detached from the frontiers of high intrinsic reward

2. Derailment : the exploratory mechanisms of the algorithm prevent it from returning to previously visited states

Detachment

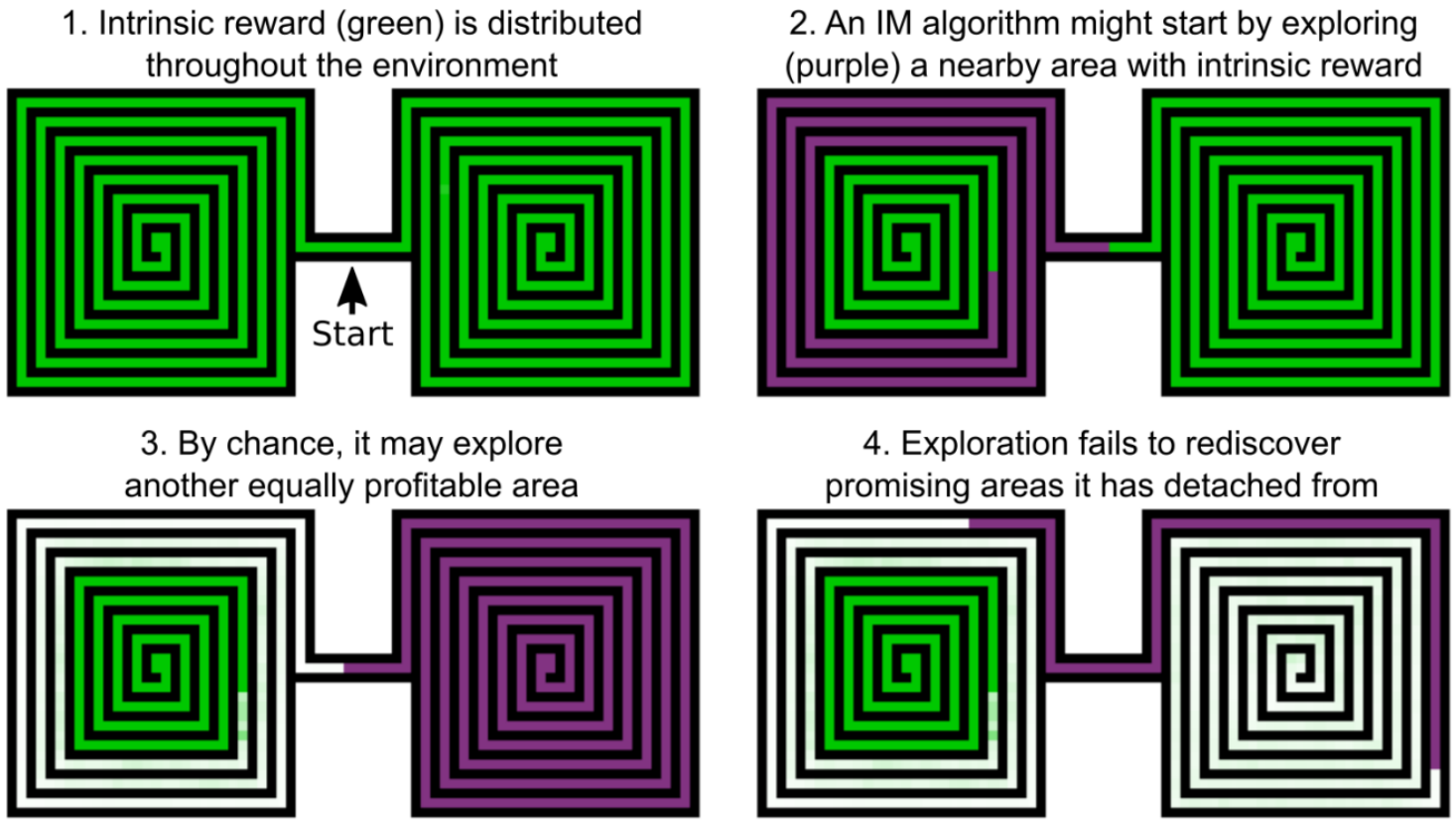


Figure 1: **A hypothetical example of detachment in intrinsic motivation (IM) algorithms.**

Derailment

Most RL algorithms:

- take promising policy, perturb it, hope it explores further
- most likely breaks policy!
    - especially as length, complexity, & predicsion of sequence increases

methods

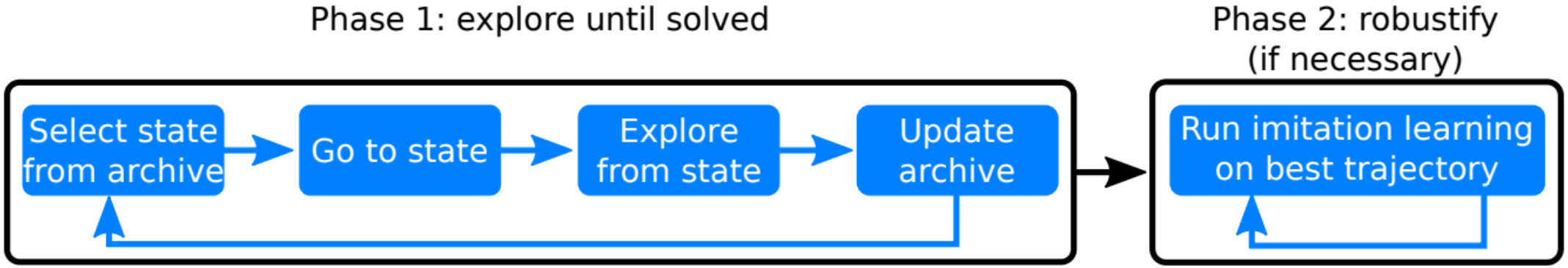
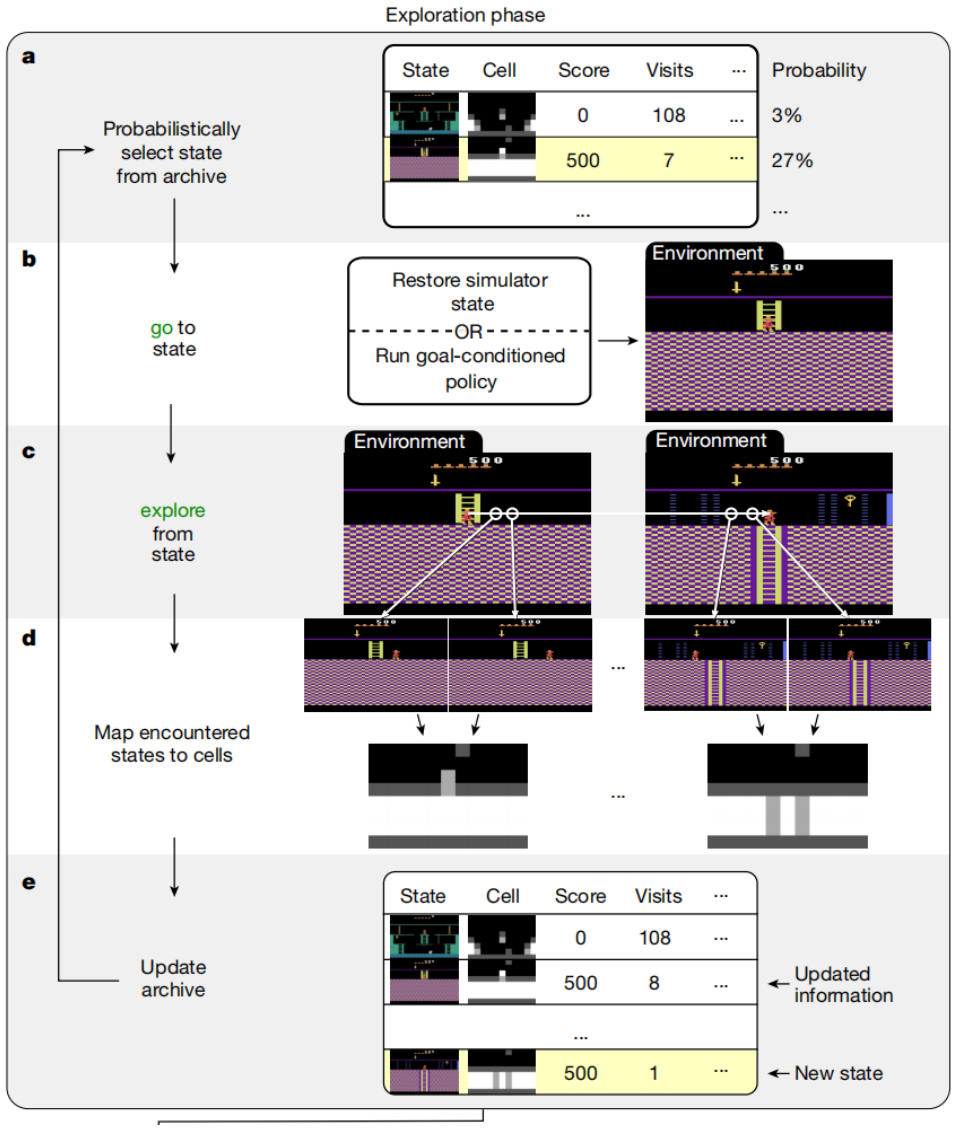
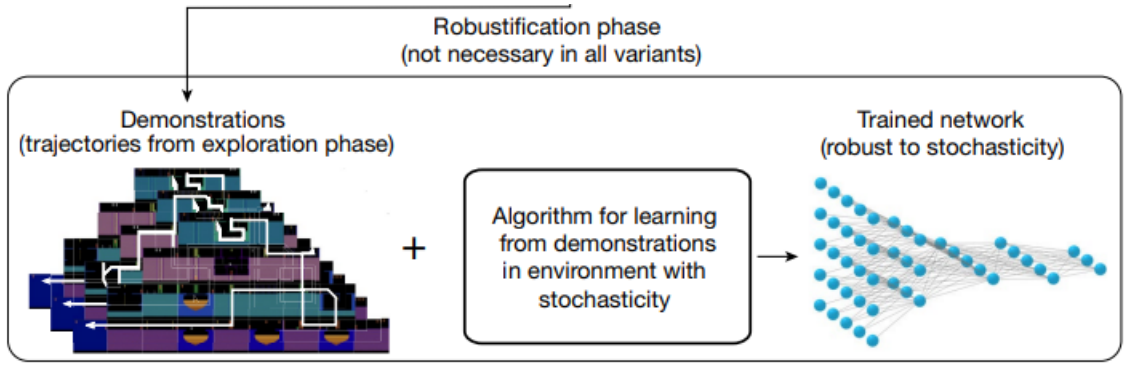


Figure 2: **A high-level overview of the Go-Explore algorithm.**

Phase 1： builds up an archive of interestingly different game states, which we call “cells”

methods


Exploration phase
a
State
Cell
Score
Visits
...
Probability
0
108
3%
500
7
27%
Probabilistically select state from archive
b
go to state
Restore simulator state
OR
Run goal-conditioned policy
Environment
c
explore from state
Environment
Environment
d
Map encountered states to cells
e
Update archive
State
Cell
Score
Visits
0
108
500
8
Updated information
500
1
New state


Robustification phase
(not necessary in all variants)
Demonstrations
(trajectories from exploration phase)
+
Algorithm for learning from demonstrations in environment with stochasticity
Trained network
(robust to stochasticity)

Phase1 - Cell representations

1. Cell representations without domain knowledge

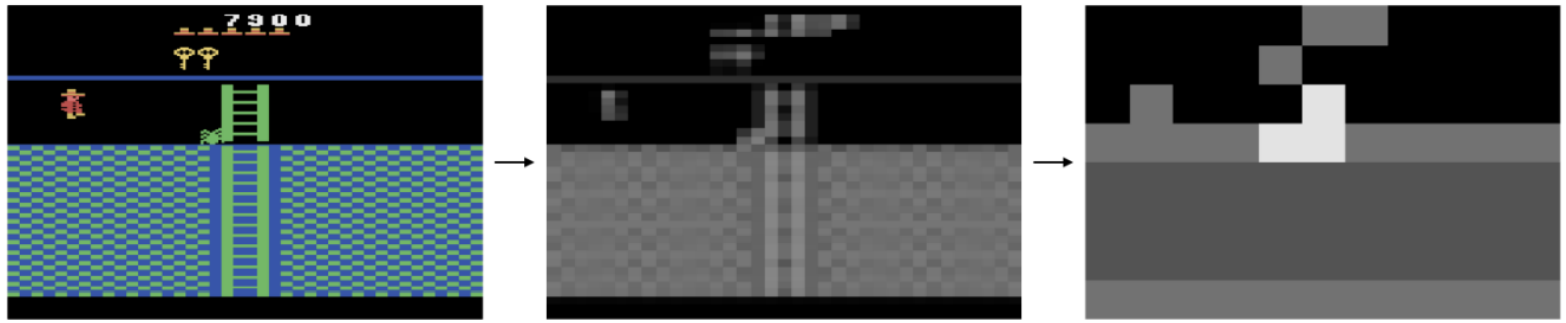


Figure 3: **Example cell representation without domain knowledge, which is simply to down-sample each game frame.** The full observable state, a color image, is converted to grayscale and downscaled to an 11×8 image with 8 possible pixel intensities.

2. Cell representations with domain knowledge(easy-to-provide domain knowledge)

Montezuma’s Revenge： (1) combinations of the x, y position of the agent (2) room number (3) level number (4) in which rooms the currently-held keys were found

Pitfall: only the x, y position of the agent and the room number were used

Phase1 - Selecting cells

(1) uniformly at random

(2) heuristic : differs depending on the problem , but at a high level, the heuristics in our work assign a positive weight to each cell that is higher for cells that are deemed more promising

The count subscore for each of these attributes is given by:

(1) 选择次数
(2) 访问次数
(3) 选择该位置找到更好的cell的次数

$$CntScore(c,a) = w_a \cdot \left(\frac{1}{v(c,a)+\varepsilon_1}\right)^{p_a} + \varepsilon_2$$

$$NeighScore(c,n) = w_n \cdot (1 - HasNeighbor(c,n))$$

$$CellProb(c) = \frac{CellScore(c)}{\sum_{c'} CellScore(c')}$$

$$LevelWeight(c) = 0.1^{MaxLevel-Level(c)}$$

$$CellScore(c) = LevelWeight(c) \cdot \left[\left(\sum_n NeighScore(c,n)\right) + \left(\sum_a CntScore(c,a)\right) + 1\right]$$

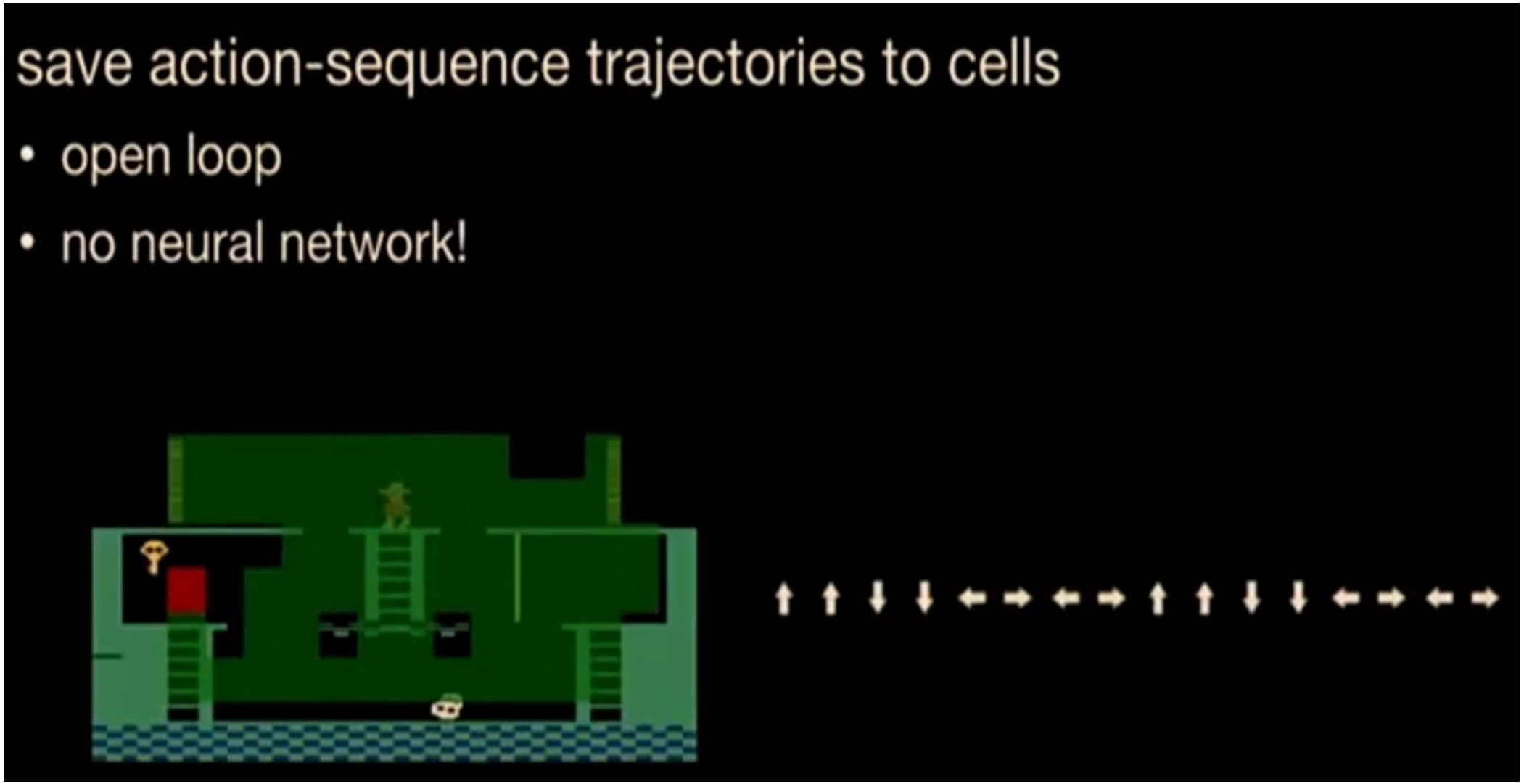
Phase1 - Returning to cells and opportunities to exploit deterministic simulators

One of the main principles of Go-Explore is to return to a promising cell without added exploration before exploring from that cell. The easiest way to return to a cell is if the world is deterministic and resettable

Two different types of problems
(1) require stochasticity at test time only
(2) require stochasticity during both testing and training

For the experiments in this paper, because we harness deterministic training, we could return to a cell by storing the sequence of actions that lead to it and subsequently replay those actions

Phase1 - Returning to cells and opportunities to exploit deterministic simulators



Phase1 - Exploration from cells

Once a cell is reached, any exploration method can be applied to find new cells. In this work the agent explores by taking random actions for k = 100 training frames, with a 95% probability of repeating the previous action at each training frame

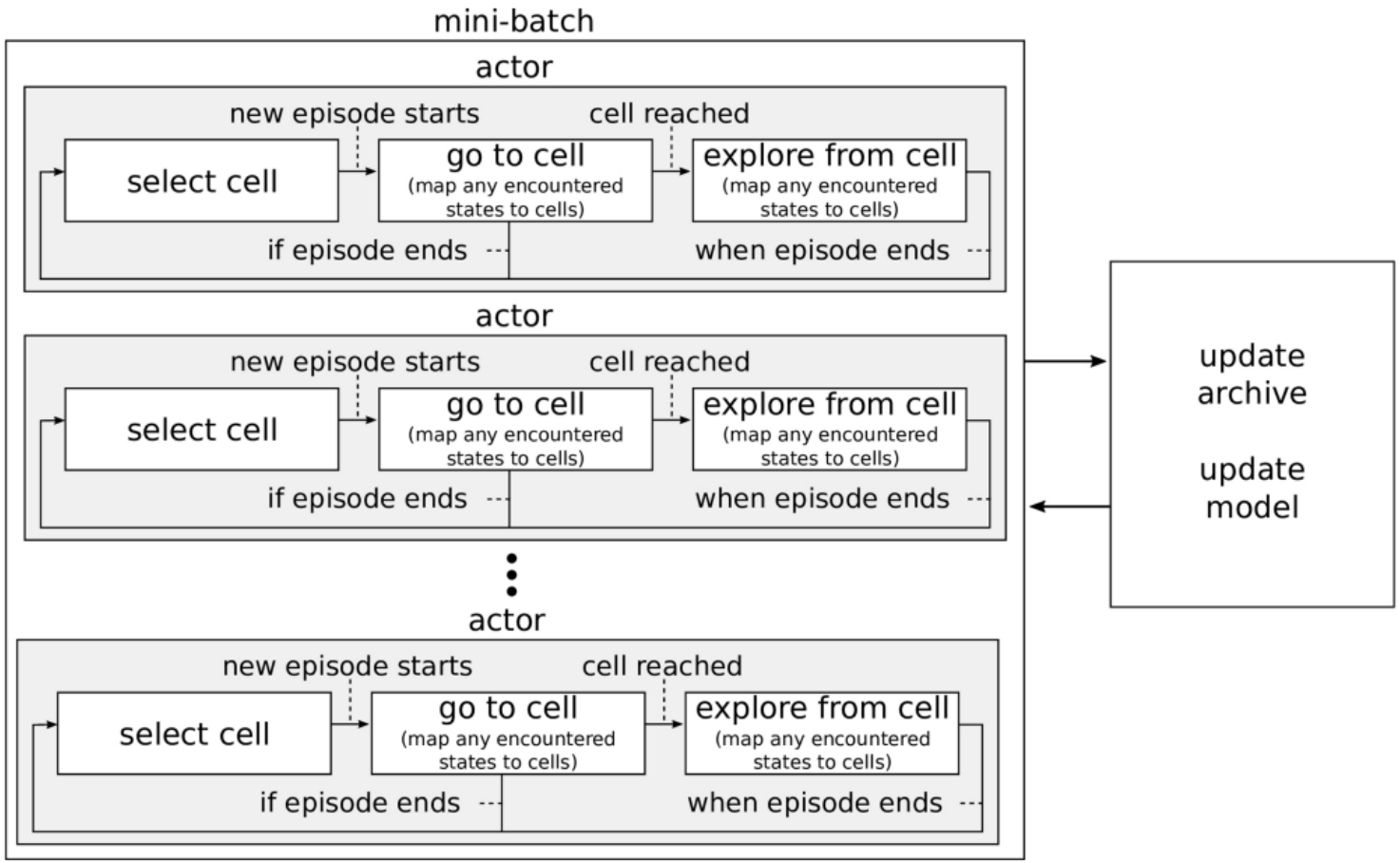
Interestingly, such exploration does not require a neural network or other controller, and indeed no neural network was used for the exploration phase (Phase 1) in any of the experiments in this paper (we do not train a neural network until Phase 2). The fact that entirely random exploration works so well highlights the surprising power of simply returning to promising cells before exploring further, though we believe exploring intelligently (e.g. via a trained policy) would likely improve our results and is an interesting avenue for future work.

Policy-based Go-Explore

The algorithm builds off the popular PPO algorithm. At the heart of policy-based Go-Explore lies a goal-conditioned policy $\pi_\theta(a|s, g)$

the agent will select a goal for the policy according to one of three rules:
(1) with 10% probability, randomly select an adjacent cell
(2) with 22.5% probability, select any adjacent cell, whether already in the archive or not
(3) in the remaining 67.5% of cases, select a cell from the archive according to the standard cell-selection weights

Policy-based Go-Explore



Extended Data Fig. 6 | Policy-based Go-Explore overview. With respect to their practical implementation, the main difference between policy-based Go-Explore and Go-Explore when restoring a simulator state is that in policy-based Go-Explore there exist separate actors that each have an internal loop switching between the 'select', 'go', and 'explore' steps, rather than one outer loop in which the 'select', 'go', and 'explore' steps are executed in synchronized batches. This structure allows policy-based Go-Explore to be easily combined with popular reinforcement learning algorithms like A3C[20], PPO[21] or DQN[15], which already divide data-gathering over many actors.

Phase1 - Updating the archive

Updates in two conditions

1. the agent visits a cell that was not yet in the archive

four associated pieces of metadata

(1) how the agent got to that cell

(2) the state of the environment at the time of discovering the cell

(3) the cumulative score of that trajectory

(4) the length of that trajectory

2. a newly-encountered trajectory is “better” than that belonging to a cell already in the archive.

notes :

(1) not reset the counter that records the number of times the cell has been visited

(2) not integrated into the trajectories of other cells

Phase1 - Batch implementation

We implemented Phase 1 in parallel to take advantage of multiple CPUs (our experiments ran on a single machine with 22 CPU cores): at each step, a batch of b cells is selected (with replacement) according to the rules described in Section 2.1.2 and Appendix A.5, and exploration from each of these cells proceeds in parallel for each. Besides using the multiple CPUs to run more instances of the environment, a high b also saves time by recomputing cell selection probabilities less frequently, which is important as this computation accounts for a significant portion of run time as the archive gets large (though this latter factor could be mitigated in other ways in the future). Because the size of b also has an indirect effect on the exploration behavior of Go-Explore (for instance, the initial state is guaranteed to be chosen b times at the very first iteration), it is in effect a hyperparameter, whose values are given in Appendix A.6.

Phase2 - Robustification

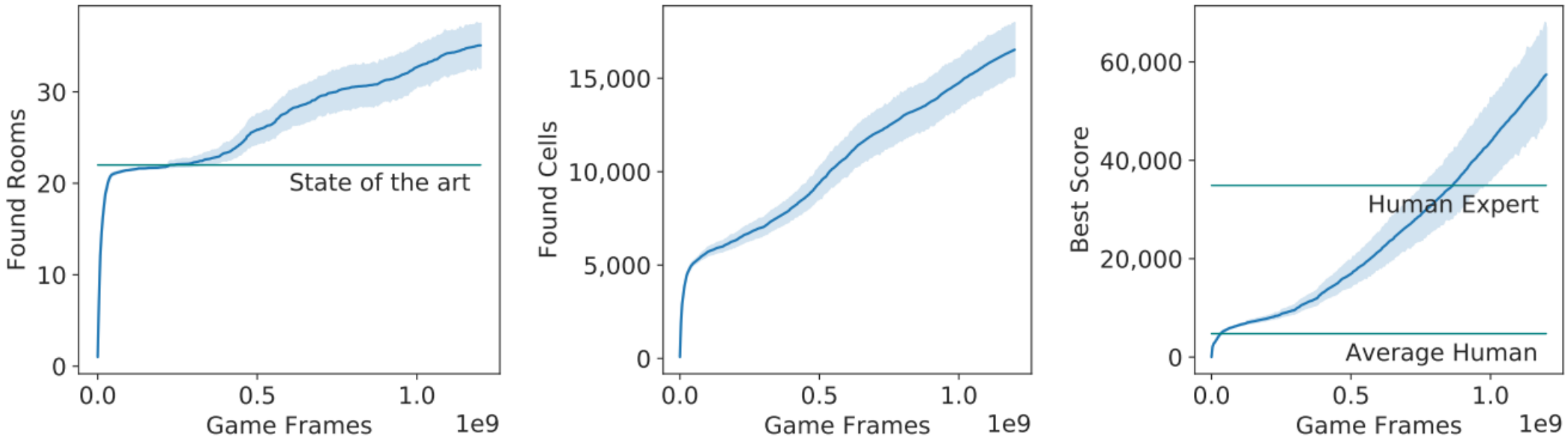
Creating a policy robust to noise via imitation learning, also called learning from demonstration (LfD)

Backward Algorithm: It works by starting the agent near the last state in the trajectory, and then running an ordinary RL algorithm from there (in this case Proximal Policy Optimization (PPO)

Additional experimental and analysis details

We introduce stochasticity into the Atari environment with two previously employed techniques: random no-ops and sticky actions

Results

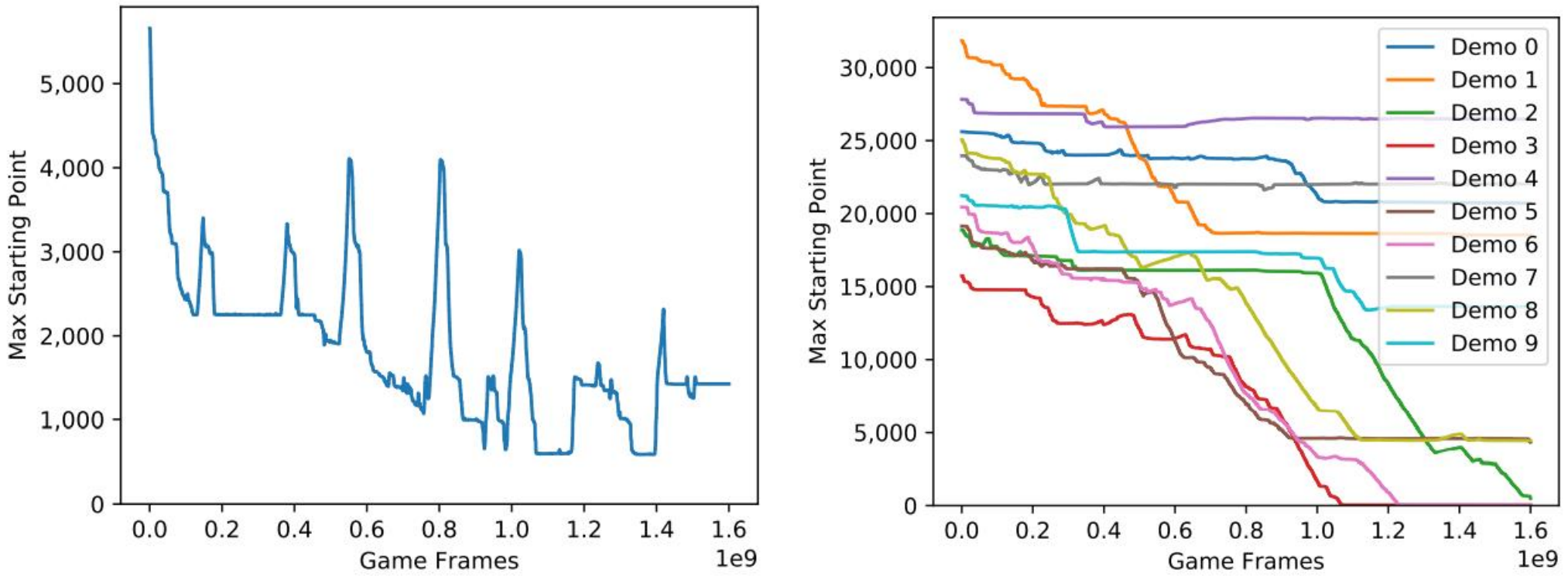


(a) Number of rooms found (b) Number of cells found (c) Maximum score in archive

Figure 4: **Performance of the exploration phase of Go-Explore with downscaled frames on Montezuma's Revenge.** Lines indicating human and the algorithmic state of the art are for comparison, but recall that the Go-Explore scores in this plot are on a deterministic version of the game (unlike the post-Phase 2 scores presented in this section).

video : https://youtu.be/civ6OOLoR-I

Results



(a) Failed robustification with 1 demonstration

(b) Successful robustification with 10 demonstrations

Figure 5: **Examples of maximum starting point over training for robustifying using different numbers of demonstrations.** Success is achieved as soon as *any* of the curves gets sufficiently close (e.g. within 50 units) to 0, because that means the agent is able to perform as well as at least one of the demonstrations.

Results

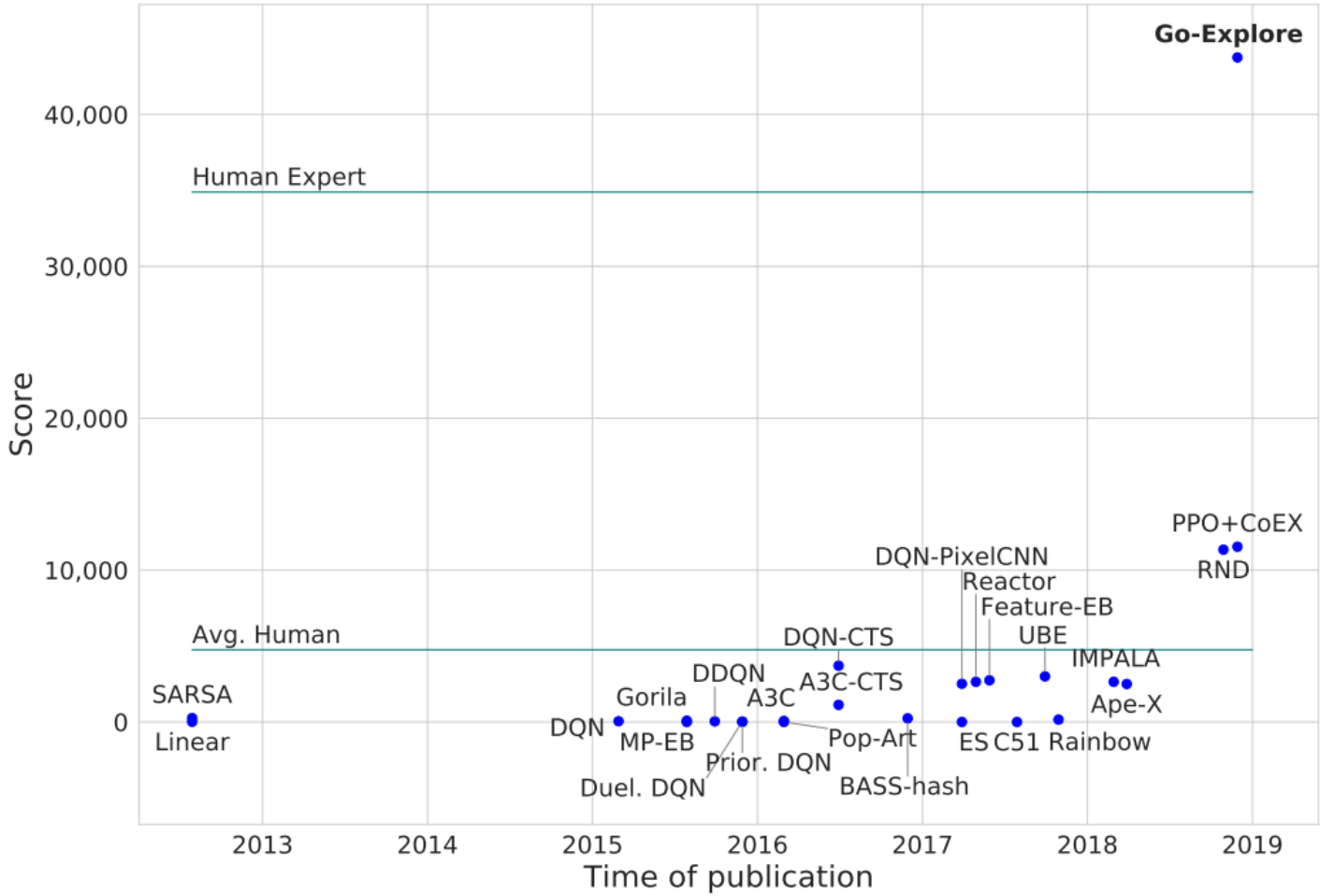
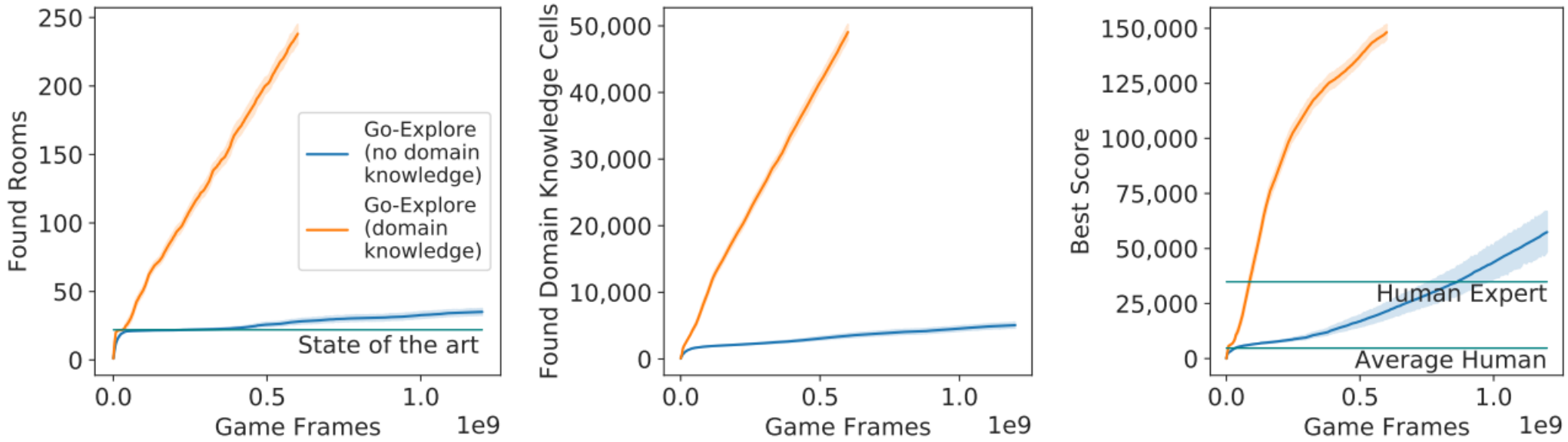


Figure 6: **History of progress on Montezuma's Revenge vs. the version of Go-Explore that does not harness domain knowledge.** Go-Explore significantly improves on the prior state of the art. These data are presented in tabular form in Appendix A.9.

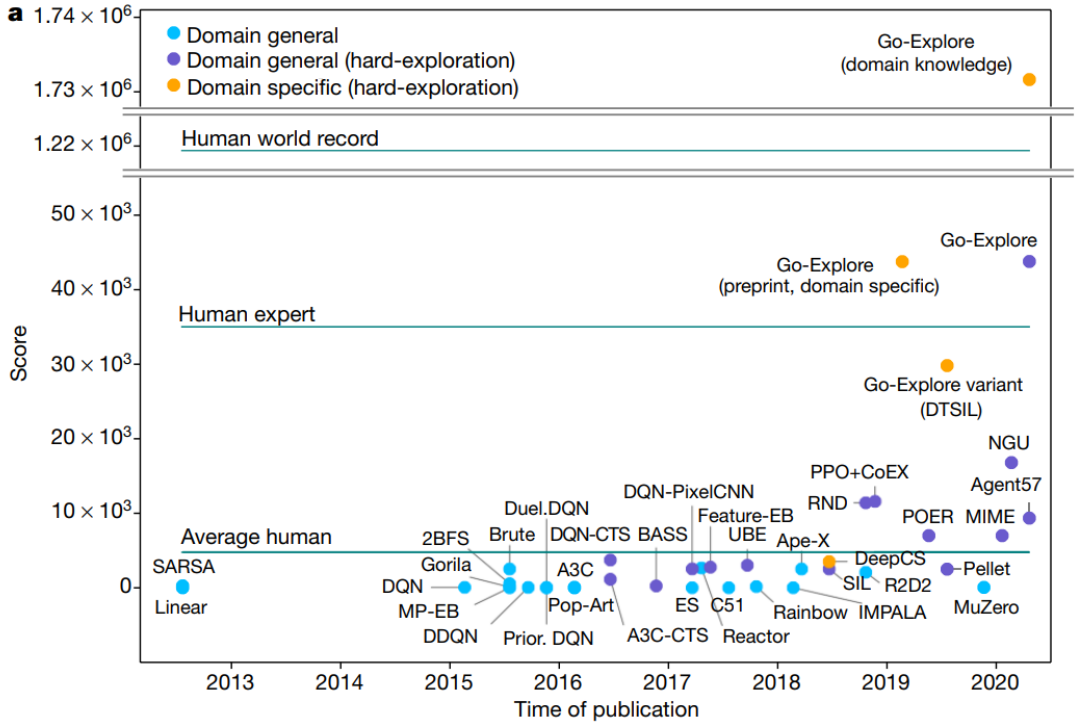
Results



(a) Number of rooms found (b) Number of cells found (c) Maximum score in archive

Figure 7: **Performance on Montezuma's Revenge of Phase 1 of Go-Explore with and without domain knowledge.** The algorithm finds more rooms, cells, and higher scores with the easily provided domain knowledge, and does so with a better sample complexity. For (b), we plot the number of cells found in the no-domain-knowledge runs according to the more intelligent cell representation from the domain-knowledge run to allow for an equal comparison.

Results



https://youtu.be/u6_Ng2oFzEY

b

Game	Exploration phase	Robustification phase	State-of-the-art performance	Average human
Berzerk	131,216	**197,376**	1,383	2,630
Bowling	247	**260**	69	160
Centipede	613,815	**1,422,628**	10,166	12,017
Freeway	34	**34**	**34**	30
Gravitar	13,385	**7,588**	3,906	3,351
Montezuma's Revenge	24,758	**43,791**	11,618	4,753
Pitfall	6,945	**6,954**	0	6,463
Private Eye	60,529	**95,756**	26,364	69,571
Skiing	4,242	**3,660**	10,386	4,336
Solaris	20,306	**19,671**	3,282	12,326
Venture	3,074	**2,281**	1,916	1,187

Results

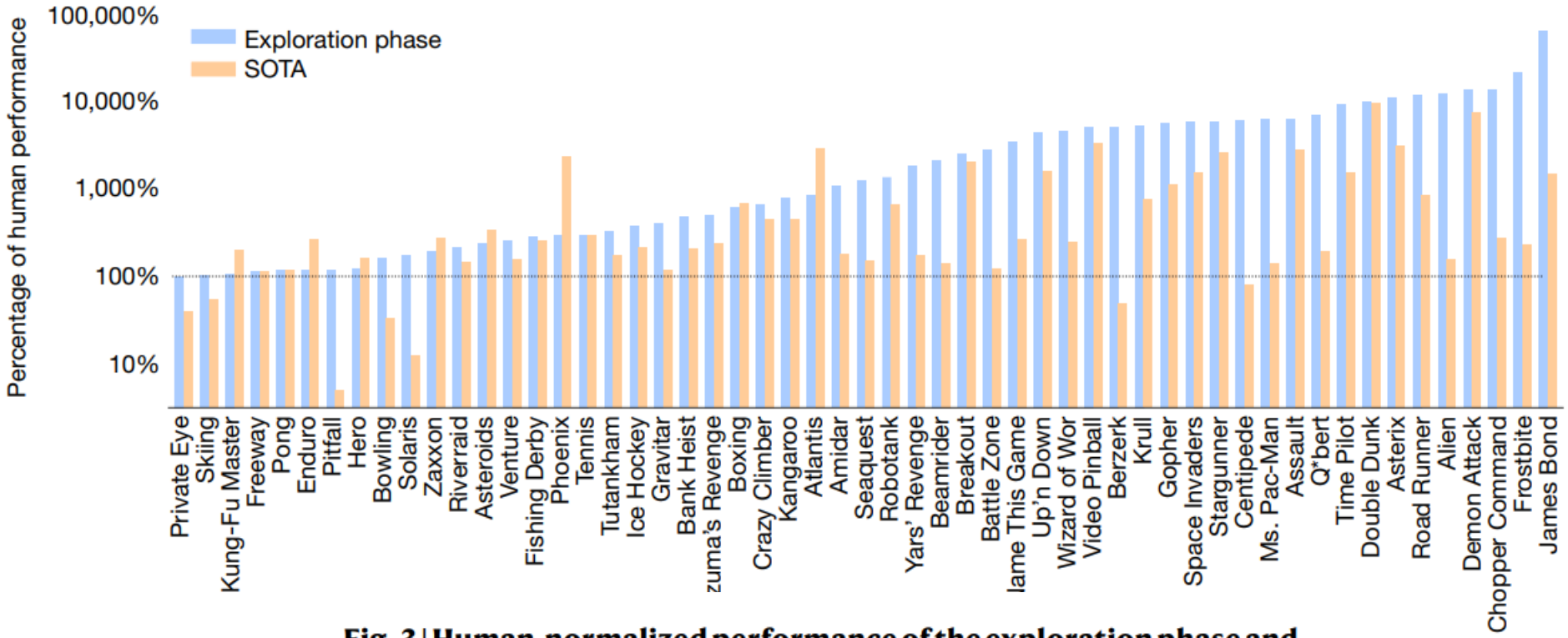


Fig. 3 | Human-normalized performance of the exploration phase and state-of-the-art algorithms on all Atari games. The exploration phase of Go-Explore exceeds average human performance in every game, often by

85.5% of these games the trajectories reach scores higher than those achieved by state-of-the-art reinforcement learning algorithms