SAPIENZA
UNIVERSITÀ DI ROMA

Improving Exploration in Sparse-Reward Environments through Reachability Bonuses

Department of Computer, Control and Management Engineering
PhD in Engineering in Computer Science (XXXIV cycle)

Jim Martin Catacora Ocana
ID number 1754293

Advisor
Prof. Daniele Nardi

Co-Advisor
Prof. Roberto Capobianco

Academic Year 2021/2022

Thesis not yet defended

Improving Exploration in Sparse-Reward Environments through Reachability Bonuses

PhD thesis. Sapienza University of Rome

© 2022 Jim Martin Catacora Ocana. All rights reserved

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: catacora@diag.uniroma1.it

Dedicated to
My beloved parents
Luz & Edwin

Abstract

Sparse-reward environments are famously challenging for deep reinforcement learning (DRL) algorithms. Several general-purpose DRL methods, that perform outstandingly otherwise, utterly fail when the incentives they receive are sparse. Nevertheless, the prospect of solving intrinsically sparse-reward tasks in an end-to-end fashion without any expert intervention is highly appealing. As that would simplify and possibly even speed up the overall process of obtaining a solution for a given problem, and it would also circumvent the assimilation of detrimental human biases in that solution. Such an aspiration has recently led to the development of countless DRL algorithms able to automatically handle sparse-reward environments to some extent. Without any of them being a definitive solution yet. Ultimately, it is this realization that end-to-end learning with sparse rewards is a purposeful and still open research problem that motivates this thesis to look deeper into such a topic. In particular, the focus of the present work is twofold. For one, to elevate the understanding of which scenarios and their features are even now out of reach for present-day techniques designed to combat reward sparsity. And for two, to engineer a novel algorithm that pushes the boundary of what can be solved with respect to its immediate predecessors. To fulfill these objectives this doctoral effort is structured into three stages. First, the composition of an insightful overview that, stemming from the reading of numerous articles proposing new methods for facing problems with sparse rewards, identifies several environmental features which decidedly impact positively or negatively the performance of such techniques. One of the many merits of this compilation is that it pinpoints quite a few holes in the existing literature. Namely, environments where better algorithms are needed as well as environments where, even more critically, further experimentation is needed since their problematics have been theorized but not yet thoroughly tested. In its second stage, seeing this latter research gap, the present thesis conducts four benchmarking studies. Each of them turns the spotlight on a particular environmental property deemed a priori difficult to handle in the absence of reinforcement and that, importantly, had not been systematically assessed up to now. In practice, every benchmarking evaluates a handful of state-of-the-art methods in one or more novel sparse-reward domains created specifically within this work to internalize exactly one of the previous features. Across the board, these studies finally demonstrate that current algorithms are not presently suitable for dealing with any of the four examined properties. However, this is all not bad news as, taking it from another perspective, these studies also pinpoint to four open problems whose solution demands fresh ideas and may lead to exciting new research directions, some of which are already discussed in this document. The last stage involves the betterment of a contemporary technique, which during the execution of the prior benchmarking endeavors was detected to have a blatant weakness in the presence of a peculiar environmental feature. In an attempt to overcome such an issue, this thesis reformulates the exploration apparatus of said algorithm and in particular its bonus scheme. At last, experiments carried out in various custom-made domains prove the success of these modifications. Not only is the improved method capable of efficiently solving tasks containing the reported property, but it also maintains the advanced benefits already provided by its precursor.

Contents

Chapter 1

Introduction

In recent years, deep reinforcement learning (DRL) has achieved remarkable results in a number of domains. For instance, AlphaGo Zero [93] learns to play Go with superhuman performance without relying on any source of expert knowledge. Indeed, it acquires this proficiency purely by playing against itself from scratch (i.e., self-play). After approximately 3 days and 5 million games of experience, it discovered winning strategies never before applied by humans. It was also able to best AlphaGo 100 matches to 0, the latter AI being famous for defeating the European Go champion in 2015. The videogame Starcraft II is another scenario in which reinforcement learning has thrived. Specifically, not long ago, AlphaStar [112], managed to beat dozens of human opponents in head-to-head man vs machine clashes. Similarly, this algorithm enforces self-play, but it also takes advantage of expert data from recorded games during training. The evaluation of AlphaStar took place within an online gaming platform. There, it competed against several human pros and eventually reached a Grandmaster level for all three StarCraft races. Gaining ratings above 99.8% of officially ranked players. Still within the realm of videogames, Agent57 [76] made a formidable breakthrough in the standard suite of the Arcade Learning Environment (ALE) [8]. It became the first learner to surpass the scores of non-expert players in all 57 Atari games composing said suite. Notably, to consummate this feat, it had to adopt a few techniques appearing in the literature on learning with sparse rewards. In particular, the usage of novelty bonuses as proposed in Burda et al. [12] was crucial for its overcoming of hard-exploration games. Robotics environments have as well partaken in these success stories. In Levine et al. [58], a real PR2 robot mastered a handful of menial activities using only one of its arms. These included: placing a hanger on a rack and screwing a cap onto a bottle. Gu et al. [36] trained a pair of physical manipulators to jointly open a door. Meanwhile, in Haarnoja et al. [37], a real-world four-legged Minitaur robot managed to learn a stable walking gait in about 2 hours.

Unfortunately, despite these accomplishments, deep reinforcement learning still underperforms in various settings; one of them being sparse-reward environments. That is, domains where rewards are zero everywhere except in few special states, which additionally lie far away from each other and from every initial state. At least sufficiently far to be near impossible for a uniform policy to travel by chance from one rewarding state to the next in a reasonable time. These scenarios are

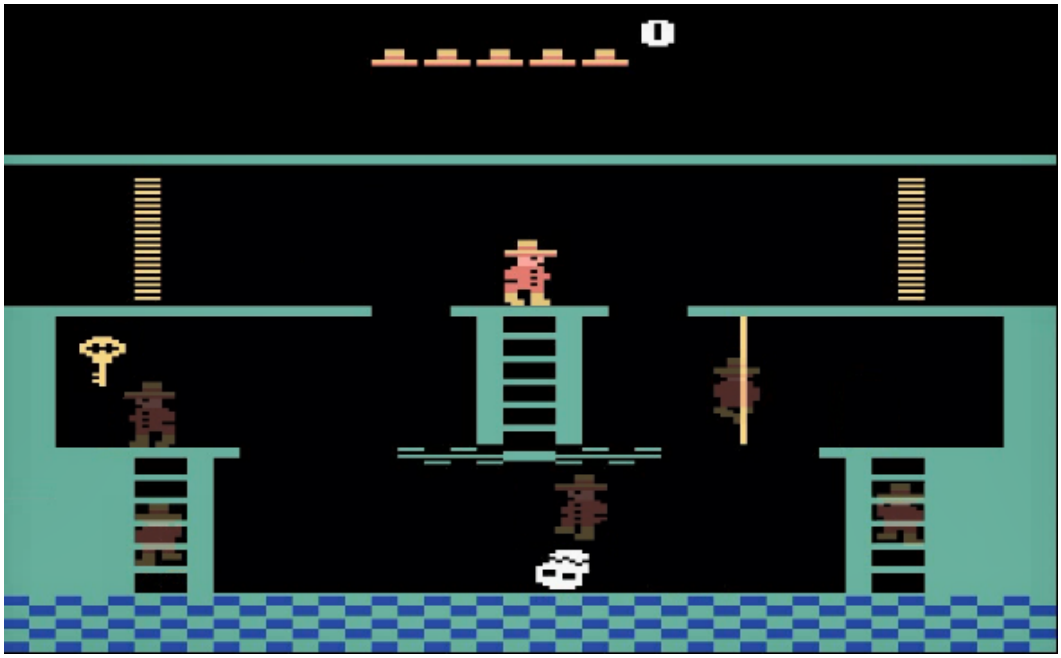


Figure 1.1. First room of Montezuma's revenge.

difficult for DRL because in absence of ready available reinforcement this technique becomes aimless. In other words, it has no reason to prefer one action over another. Ultimately, such a lack of motivation pushes DRL to settle on trying every action evenly at random. A strategy that leads nowhere in large and complex environments. An infamous real-life sparse-reward problem is the Atari game Montezuma's revenge (MR). In the seminal work of Mnih et al. [65], this was one of the few games for which DQN showed no competence at all. In fact, this learner never scored above zero throughout its training. As seen in Figure 1.1, in the first room of this game the avatar is required to perform an elaborate sequence of actions before reaching the key on the left. In sequential order, it must jump over an opening in the floor, climb down a rope, go down a ladder, evade an enemy, go up another ladder and finally jump up to grab said key. Crucially, getting this item grants the first (positive) reward in this room. Several state-of-the-art end-to-end DRL algorithms [42, 22, 43, 47, 86] have utterly failed to solve MR when learning from images. Scoring much lower than 4,367 points, which is the average for non-expert players [65]. A debacle that comes in spite of these very methods achieving impressive results in most other games within the ALE environment. For instance, Ape-X DQN [43] reports a human-normalized median score across 57 Atari games of 434% (with no-op starts). This means that if we chose to play a game at random, Ape-X DQN is likely to earn over four times as many points as an untrained human competitor. Yet, after 22,800 million frames of experience, it is still unable to complete the first level of MR, obtaining just 2,500 points on average.

Sparse-reward environments are not just a triviality arising in videogames where rewards are inherently provided by the system. But in fact there are many scenarios in which it would be best to formulate the reinforcement learning problem in such

a sparse fashion. This is true even for real-world applications where the reward function could be specified by a person. Generally speaking, referring to the task for which we wish to develop an autonomous agent, a first thing to know is if it has already been solved by humans. When such is true, often the most effective way to proceed is to collect a sufficiently large database of demonstrations executed by experts. Having done that, we can then cast the problem under the frameworks of imitation learning or inverse reinforcement learning. However, this may not always be possible or desirable. Sometimes, it may be unfeasible to gather enough demonstrations, as this could be too costly or time-consuming. Other times, we may face a task for which people are incapable of providing just one satisfactory solution. And there can even be times when we may prefer not to limit our learner to the human-level intelligence entrap within such demonstrations. If we found ourselves in one of those cases, then we should consider other learning alternatives. At this point, a mostly automatized learning framework that has recently rise to popularity is the one proposed by Ecoffet et al. [21]. This is a non-DRL approach that banks on planning for exploring the entirety of an environment and then applies imitation learning for developing a generalizing policy. This idea has obtained the best machine learning result to date in Montezuma's revenge (40K points without prior domain knowledge); thus, it is certainly promising. However, such an approach only works well in perfectly deterministic environments, a fact that severely limits its applicability. It is true that the authors of this novel framework have also put forward a version of it that is confined within the boundaries of DRL, but this has not been proven quite as successful as its original proposal. Hence, we cannot depend on it just yet. This means that we will still be in problem if we are faced with stochastic environments, which are the norm in the real-world. Once in this situation, an obvious paradigm to pursue due to its simplicity is to manually define a reward function; an approach called reward engineering. In actuality, this is a common practice in which researchers define dense reward signals based on their own ingenuity. For instance, they may need to devise a continuous function that correctly quantifies the proximity of any state to its closest goal. Or, they could specify a set of subgoal states that serve as stepping stones along a path leading to a terminal goal. At any rate, the basic idea is to structure these hand-crafted rewards in such a way that as the agent accumulates more of them, it gets closer to the desired goal. Obviously, these extra incentives augment the binary success-failure nature of the reinforcement underlying most problems; and hence, they effectively eradicate reward sparsity. Sadly, since reward engineering reshapes the original reward function, it could lead to some unwanted outcomes. Namely, the creation of new local minima, final policies that are suboptimal in the unaltered problem and reward hacking. The latter term denotes this situation when the agent's resulting behavior is not what the designer intended. For example, wandering around collecting points instead of finishing a race [15]. Furthermore, reward engineering as well as other techniques that insert human expertise, e.g., advice guidance, manual curriculum or manual task decomposition exhibit one other major shortcoming. They can hardly handle highly-complex tasks, where human intuition is not enough for sketching a favorable path towards a solution. It is finally in such cases, where reward engineering also fails to meet our needs, that we have no choice but to fully embrace the innate reward sparsity of an environment. To summarize, these situations in which we

ought or may even prefer to work with sparse rewards under an end-to-end DRL framework involve: i) having no access to demonstrations, ii) engaging stochastic environments, iii) undertaking a task too perplexing for people, iv) aiming at true optimality and/or v) striving for superhuman performance.

Upon recognizing this necessity for handling sparse-reward environments through automatic algorithms. That is, ones that can do without any aid from experts throughout their learning process (i.e., end-to-end). DRL researchers speedily took on this challenge and have ever since been investigating a wide range of avenues to address the issue of reward sparsity. Some of these miscellaneous ideas include the following. Using dense interestingness measures such as novelty, curiosity, empowerment, compressiveness and many others as intrinsic motivation [9, 74, 98, 12, 84, 49]. Optimizing a set of exploration policies to cover the entire state space [57]. Taking approximations to Bayesian uncertainty as guidance for deep exploration [72, 69, 90]. Exploiting the behavioral diversity found across an array of policies [23]. Leveraging hindsight to learn from mistakes in multi-goal scenarios [2]. Imitating an agent's own past profitable or unfamiliar trajectories [70]. Automatically generating a curriculum of subtasks whose difficulty increases to match the agent's current expertise [29, 28, 100]. And imposing a hierarchy of learners that communicates through acquired subgoals [111]. In the end, these efforts have paid off tremendously. Indeed, taking Montezuma's revenge as reference, several end-to-end approaches have by now managed to complete the entire first level of this game directly from pixel input. This corresponds to navigating up to 24 rooms and scoring over 10,000 points. In other words, more than double of what an untrained human player earns. Random network distillation (RND) [12], created with the sole mission of besting sparse-reward tasks, was the first of such methods to reach this milestone. Using intrinsically generated novelty bonuses that constantly drive exploration toward never-before-seen areas of the world, this algorithm obtained around 11,000 points in MR. Then, a couple of years later, general-purpose techniques such as Never Give Up (NGU) [77] and Agent57 [76] followed suit, mastering Montezuma's revenge to a similar degree as RND. Those two methods employ under the hood advanced DRL algorithms, which by themselves already produce state-of-the-art performances in domains with dense rewards. But interestingly, to get superhuman scores in less informative games rewardwise (e.g., MR), they adopt the very bonuses formulated by RND. Because these significantly enhance the exploration capabilities of said cutting-edge techniques.

Although present-day automatic algorithms have yielded the exciting results recounted previously, and are staggeringly more competent than their predecessors in outclassing reward sparsity. They are nonetheless far from being the end of the road in this area of research. For instance, even with respect to Montezuma's revenge, their effectiveness still lags woefully behind the human world record of 1.3M points [3]. On top of that, another lingering issue is their lack of generality. As there have been many reports of them becoming absolutely incompetent in the presence of certain unassuming features within an environment. Case in point, in [49], it is disclosed that RND secures 0 points in MR when the observations it receives contain pixel-wise noise in quantities that would not trouble a person. In this context, seeing that tackling sparse rewards with end-to-end DRL promises great benefits and that there is still plenty of room for improvement, the present thesis makes the decision to

center itself around this area of research. This work starts by taking a closer look into which environmental properties have a direct and pronounced impact on state-of-the-art approaches, positive or negative. To this end, an extensive review of the literature is conducted that compiles numerous such features. Specially those that according to previous works are deemed to influence learning under sparse rewards. Such a compilation gathers features that have been exploited in the past as well as those reported to be troublesome. A notable revelation arising from said analysis is that presently the effects of some of those examined features have not been thoroughly and systematically studied. Not only that, but various other challenging properties left out of the previous review have barely been paid attention in the literature. For that reason, this thesis further performs four benchmarking studies, each focused on one demanding yet stimulating feature. In each examination, state-of-the-art algorithms designed to deal with sparse rewards are tested on new domains created to highlight a particular environmental property. A first benchmarking is centered on distractors (i.e., elements that an agent can interact with but are irrelevant to the resolution of a task). In it, two motivating new sparse-reward scenarios are put forward containing the so-far largely overlooked class of exploration-intensive distractors. Its results show that present methods become too fixated on such inessential elements. So much so, that they end up interacting solely with them while completely ignoring the task they were expected to solve. The second benchmarking examines tasks that demand the re-exploration of the environment. In particular, a novel maze displaying such need is fabricated, which asks the interacting agent to bring back to its starting position an item found at a remote site. A journey in which, because of the peculiar layout of the maze, said agent has no choice but to visit every location twice. Tests on this domain clearly demonstrate that present approaches are not motivated to retrace their steps. Because during exploration they cannot differentiate between first and subsequent visits to a given place within the same episode. As a result, they are only capable of mastering the outbound portion of the aforementioned journey. The third benchmarking experiments with an environment where high-level skills (i.e. composed of multiple atomic actions) must be reused several times within the same task. In this study, an indicator is also devised, which puts a number on how much methods developed to tackle reward sparsity leverage reusability to improve their sample efficiency. It is finally reported that, according to such indicator, current algorithms learn from scratch each instance of a skill (i.e., they fail to generalize). And this makes them severely data-hungry during exploration. In the fourth benchmarking, a synchronous dual-arm manipulation task is implemented. Its solution involves opening a regular-size padlock with the help of a regular-size key. Although this problem seems trivial from a human's perspective, the high precision required to complete it confers it a very special property. That is, any sensible compression of the state space will irrevocably end up with a tremendous amount of abstract regions. Here, sensible means to split the space into chunks small enough to guarantee that at least one of them always corresponds to the task being solved. Findings from the evaluation done in such a domain suggest it may remain intractable for state-of-the-art algorithms for the foreseeable future. Abstract states are simply too many to all be explored within a reasonable time with current computing power. In summary, the outcomes of all four benchmarkings have in common that contemporary techniques, although

successful at handling sparse rewards in other domains, are not suitable for the environments proposed within these studies. To everyone else this may sound bad, but to researchers, this should be taken as an opportunity. The reality is that the results of these examinations simultaneously reveal that each of the four analyzed environmental features marks a different open problem within the field. Problems that, by no means, are necessarily intractable, but whose resolution will certainly require innovative ideas. It is worth emphasizing, that within each study, this thesis already exteriorizes insightful discussions on realistic new research directions that may precipitate the vanquishing of such unresolved issues. And the contributions of these benchmarkings do not end here, every novel environment designed within the scope of such enterprises is made freely available to the community. In this way, eager researchers could readily start testing their fresh proposals without wasting any time. For the culmination of this doctoral work, a new algorithm is proposed, which resolves a weakness observed in an earlier solution. Specifically, a tangential result emerging from the prior benchmarking efforts is the detection of a fundamental flaw in one of the methods being assessed; namely, episodic curiosity through reachability (EC). Without going into details, this algorithm crumbles in the presence of bifurcations where the branch leading to the goal is populated with plentiful deadly areas while the others have none. In that scenario, EC ends up developing an insurmountable aversion towards the murderous path, which forever prevents it from reaching the goal. To solve this, the present thesis proposes the reformulation of the bonus computation carried out by EC together with the introduction of a well-structured backtracking bonus. These modifications create a novel algorithm that in theory is strongly motivated to methodically navigate along every single branch of an environment within an episode, regardless of its deadliness. A series of experiments finally validate the effectiveness of this method. They unequivocally show that it is indeed capable of eventually finding the goal in environments housing treacherous paths. Plus, they do not uncover any perceivable detrimental side-effects that would lower its performance compared to its predecessor in other domains.

1.1 Thesis Layout

This document has been divided into 10 chapters whose contents are briefly described next.

Chapter 1 presents the topic of learning with sparse rewards, which is the focus of this thesis. It quickly defines this issue, explains why it is important and chronicles its historical development. The chapter culminates by communicating the multiple research efforts carried out within the scope of this doctoral work and how they are link to each other.

Chapter 2 delivers a myriad of general definitions and summaries of foundational techniques regarding classical and deep reinforcement learning, which will be useful throughout the remainder of the document.

Chapter 3 provides a more precise explanation, grounded on the concepts established in the previous chapter, of why reward sparsity is inherently a difficult problem in reinforcement learning. Moreover, it expands greatly on the struggles and successes of numerous previous works at dealing with sparse rewards, while

pinpointing the key ideas that have led to progress in the subject.

Chapter 4 elaborates a thorough overview of the environmental features that even now heavily affect most of the algorithms that up to this date have been developed to tackle reward sparsity, which were priorly introduced in the preceding chapter. For each feature, it is offered: a definition, examples of methods that thrive or fail only when such a property is present and a detailed dissection of the underlying mechanisms causing such an interdependence.

The next four chapters perform four different benchmarking studies, each centered around a potentially troublesome environmental feature thus far not well understood in the literature. The first two examinations conduct a deeper investigation of two of the properties included in the former overview. Meanwhile, with respect to the other two features, though these are general ideas that have been floating around, this is the first work that truly scrutinizes them in an exhaustive and systematic manner in the context of learning with sparse rewards. Each study creates one or more new tasks that highlight a single environmental property, tests existing techniques in those domains and judiciously discusses future research directions that may finally lead to the principled conquest of such tasks.

Chapter 5 executes a benchmarking that probes exploration-intensive distractors. The latter are encompassed by the general concept of distractors, which was illustrated in a subsection of Chapter 4, However, this thesis formulates an entirely new class of distractors whose exigencies are decidedly different from those of others that have been examined before.

Chapter 6 completes a benchmarking focused on environments that to be solved demand the interacting agent to retrace its steps. This problematic has been disclosed before in the literature, as recounted in a subsection of Chapter 4. Yet, the previous known analysis was very limited; hence, the present work seeks to carry out a more rigorous assessment where a larger number of state-of-the-art methods are tested in a domain designed to better display the aforementioned demand.

Chapter 7 performs a benchmarking related to the benefit of reusing complex skills to explore certain environments more efficiently. This is an ability here noticed to be lacking in every algorithm developed to handle reward sparsity. Thus, this novel study is intended to divulge such a missed opportunity so that more researchers will pay attention to it.

Chapter 8 conducts a benchmarking that transpires in an unassuming dual-arm manipulation task, specifically crafted to possess the property that any sensible unbiased abstraction of its state space still produces an astronomical number of macro states. This never-before-analyzed feature seems a good way to wrap up this benchmarking effort as it exposes the limits of present-day techniques while dealing with sparse rewards. Not from a conceptual viewpoint, but instead from the more mundane perspective of computing power.

Chapter 9 develops a new algorithm that operates comfortably in sparse-reward environments and which makes a noticeable improvement with respect to a preceding work. This chapter describes the inefficacies identified in past solutions, explains in great detail the formulation of the new method and reports the results of various experiments that demonstrate the success of this conceptualization.

Chapter 10 encompasses the conclusions originating from the prior six chapters.

It must be noted that Chapters 4 to 9 compile six separate research papers

written during this doctoral endeavor; each chapter accommodates exactly one article. Of them, the paper giving rise to Chapter 5 has been published in AIxIA 2021 with the title: Exploration-Intensive Distractors: Two Environment Proposals and a Benchmarking. In addition, the overview found in Chapter 4 has been recently resubmitted to the Journal of Artificial Intelligence Research (JAIR) by suggestion of the editors, who requested to do so after sorting out a few minor concerns. Regarding the remaining chapters, principally due to time constraints, their corresponding reports have not been put forward for publication in any venue as of yet. Nevertheless, since they manifest valuable insights and novel results, attempts to publish will be made in the coming months.

Chapter 2

Background

This chapter lays out all the fundamental concepts required for a clear understanding of the following chapters. It delves into the basics of reinforcement learning, deep neural networks and deep reinforcement learning. Moreover, it needs be pointed out that various sections here take as principal reference the introductory book written by Sutton and Barton [102].

2.1 Markov Decision Processes

MDPs [45] are a formalism for modeling sequential decision problems involving autonomous agents and the environments they inhabit. Notationally, an MDP is a tuple $\langle S, A, \mathcal{T}, \mathcal{R}, \gamma \rangle$, comprising:

- The set S of all states an environment can assume, which includes the set $S_0 \subset S$ of initial states and the set $S_G \subset S$ of terminal states,
- The set A containing every action a learning agent can execute,
- A probability distribution over transitions $\mathcal{T} : S \times A \times S \to [0, 1]$ that determines the next state given the current state and action,
- A reward function $\mathcal{R} : S \times A \times S \to R \subset \mathbb{R}$ that quantifies with a real scalar value the immediate goodness of a transition,
- And a discount factor γ that weights down the value of future rewards.

According to the former definitions, an MDP encodes concisely the interactions between an agent and its environment, which are to be interpreted as follows (see Figure 2.1). Before an episode starts, a state is drawn from S_0 and the agent is set to it. Following that, at any time step t, the agent is allowed to observe the current state of the world S_t and based on this information is expected to select an available action A_t. Once such an action has been specified, the next time step $t + 1$ is simulated. This means that, in compliance with $\mathcal{T}$ and $\mathcal{R}$, the environment transitions probabilistically to its next state S_{t+1} and the agent is analogously granted an immediate reward R_{t+1}. This cycle then repeats until the agent arrives at a terminal state (i.e., one belonging to S_G).

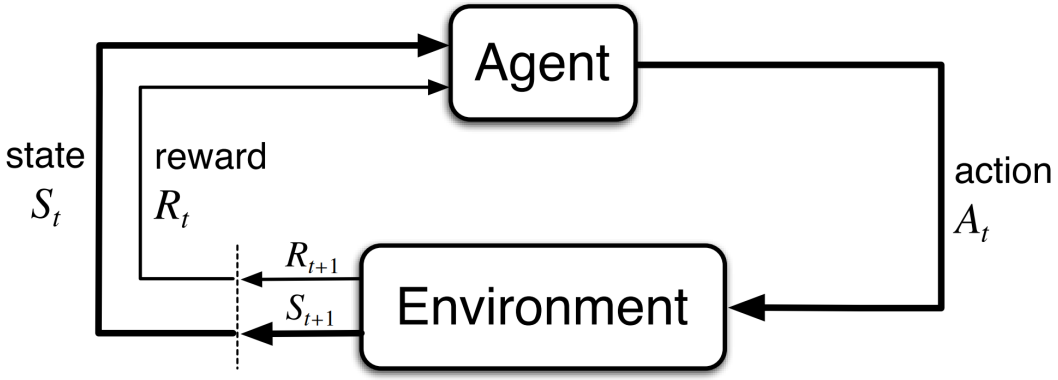


Figure 2.1. Agent-environment interaction represented by an MDP.

Note that, to be called an MDP, a decision process must additionally satisfy the Markov property [102]. This states that the next state and immediate reward depend solely on the current state and action. In other words, those two variables must be independent of information from previous time steps. To put it in mathematical terms, the following equality needs to hold true for all pairings of $r \in R$ and $s' \in S$ that are feasible within an environment:

$$\Pr\{R_{t+1}{=}r, S_{t+1}{=}s' | S_t, A_t\} = \Pr\{R_{t+1}{=}r, S_{t+1}{=}s' | S_0, A_0, R_0, ..., S_t, A_t\} \quad (2.1)$$

Unfortunately, it is not uncommon for an agent to perceive only a small region of its environment at a time, i.e., it has partial observability. In those cases, the prior Markov condition is not met, and thus, we cannot formulate the problem as an MDP. Yet, we still have the possibility to frame it within the closely-related formalization of a partially observable MDP (POMDP), which is expressed as the tuple $\langle S, A, O, \mathcal{T}, \mathcal{R}, \gamma \rangle$. There: O is the set of all observations, $\mathcal{T}$ is a probability distribution over transitions $\mathcal{T} : S \times O \times A \times S \times O \to [0, 1]$ that determines the next state and next observation given the current state and action, while $S, A, \mathcal{R}, \gamma$ maintain the definitions given for an MDP.

Within an MDP, we can also establish a policy $\pi(a \in A | s \in S)$, which is a generally stochastic mapping from states to actions that fully encodes the decision-making behavior of an autonomous agent. A similar concept also exist in a POMDP; however, because this formulation does not possess the Markov property, it is best here to be more general and instead define a policy as a mapping from past observations, actions and rewards to the present action.

2.2 The Reinforcement Learning Problem

Reinforcement learning (RL) is concerned with learning in MDPs without having an a priori model of the domain (transition and reward functions). The main characteristics of RL is that learning is accomplished through raw experience by executing actions and receiving rewards; and that learning is synonymous with the maximization of rewards collected over time [102]. The most common objective of RL is to find at least one of possibly many optimal policies π^* for the underlying

MDP. Where, an optimal policy is one that in every situation yields the highest sum of rewards, usually defined in terms of the expected cumulative discounted reward (also known as expected return).

Formally, the return G associated with a single rollout in which an agent is in state s after t time steps and thenceforth enacts policy π to receive a sequence of future rewards $R_{t+1}, R_{t+2}, ...$, is expressed as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0} \gamma^k R_{t+k+1} \tag{2.2}$$

Naturally, the corresponding expected return is computed as the expectation $E[\cdot]$ over all possible rollouts involving policy π and state s, as it is seen next:

$$E_\pi[G_t|S_t = s] = E_\pi\Big[\sum_{k=0} \gamma^k R_{t+k+1}|S_t = s\Big] \tag{2.3}$$

The term $E_\pi[G_t|S_t = s]$ is also called the value of s under π. A function $v_\pi(s)$ that encodes this information for every state in S is known as a value function. Given the previous definitions and denoting the space of all plausible policies as Π, an optimal policy π^* is any one whose values are greater than or equal to those of every other policy for every state, that is: $v_{\pi^*}(s) \geq v_\pi(s), \forall s \in S \wedge \forall \pi \in \Pi$.

The former notion of value can also be extended to every state-action pair in $S \times A$. This produces an extremely helpful function called an action-value or Q-function, which is formalized as:

$$q_\pi(s, a) = E_\pi\Big[\sum_{k=0} \gamma^k R_{t+k+1}|S_t = s, A_t = a\Big] \tag{2.4}$$

2.3 General Schemes for Learning an Optimal Policy

Approaches for solving reinforcement learning problems differ in how they address the previous optimization problem, i.e. how they extract or approximate an optimal policy. A few broad methodologies include: temporal difference [101], policy gradient [115, 103] and actor-critic [50].

2.3.1 Temporal Difference (TD)

TD methods follow a value-based approach, that is, they usually learn an action-value function, but they do not maintain any explicit model representing a policy. Instead, during evaluation, the agent selects an action greedily based on the values predicted by the acquired Q-function. Meanwhile, during training, these techniques enforce some exploration strategy, which assign non-greedy actions non-zero probabilities of being chosen.

TD algorithms learn the action-value function Q linked to the current policy π through iterative updates. Each update tries to increase the accuracy of Q by moving its current estimates closer to some targets, which ideally should be the true expected cumulative rewards tied to every state-action pair. Sadly, these expectations are unknown in RL problems; thus, TD methods have to approximate them from samples. Without further improvements, in practice this would mean

1: Initialize $Q(s,a) \leftarrow 0, \forall s \in S, \forall a \in A$
2: Observe current state S_0
3: For each timestep $t = 1, \ldots, T$ (until S_t is a final state)
4: Choose an action A_t using an exploration policy derived from Q given S_t
5: Execute action A_t
6: Observe next state S_{t+1}
7: Collect immediate reward R_{t+1}
8: Choose an action A_{t+1} using an exploration policy derived from Q given S_{t+1}
9: Update Q-value as follows: $Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1})]$
10: Set $S_t \leftarrow S_{t+1}$
10: Optimal policy: $\pi^*(s) = \text{argmax}_{a \in A} Q(s,a)$

Figure 2.2. SARSA algorithm.

to wait until each episode terminates and only then evaluate the summation seen in Equation 2.2 for every state and concurrent action pertaining to the followed trajectory. By repeating this procedure across several episodes, the resulting dataset to be used for updating Q would likely comprise multiple targets associated with the same state-action duplet. And these, taken as a whole, would give a sense of the expectation of the cumulative reward related to that pair. However, a key insight of the temporal difference framework is to apply bootstrapping, which in essence means that the previous targets are in part estimated from predictions made by some late iteration of Q. In other words, within the aforementioned sum, rewards collected from the environment are employed as usual for summands up to a time k, while the rest are approximated with Q-values. The benefit of all this is that waiting for an episode to end is not longer necessary. At this point, various TD techniques differ from each other mainly on how they implement the latter computation.

For example, SARSA is an on-policy algorithm, which means that the policy it queries during bootstrapping is the very same it has been using lately while gathering experiences from its interaction with the environment. Such a policy, oftentimes called behavior or exploration policy, normally combines the greedy strategy implicitly contained in the current estimate of Q with whichever exploration strategy adopted whilst training. Distinctively, SARSA makes a lookahead of exactly one step into the future, meaning that within its bootstrapping procedure it sets $k = 0$. As a result, after collecting the experience $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$, this method can immediately compute the following target, where A_{t+1} could potentially be a non-greedy action picked by the current exploration policy:

$$Q_{target}(S_t, A_t) = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) \tag{2.5}$$

Another influential TD method is Q-learning. Similar to SARSA, this technique also looks a single step ahead into the future as it bootstraps the estimation of its targets. But unlike the former, Q-learning is an off-policy algorithm. That is, it does bootstrapping without invoking the exploration policy, and instead, it chooses actions according to the greedy policy derived from the present iteration of Q. Consequently, given the experience example $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$, Q-learning produces the target seen next:

1: Initialize $Q(s,a) \leftarrow 0, \forall s \in S, \forall a \in A$
2: Observe current state S_0
3: For each timestep $t = 1, \ldots, T$ (until S_t is a final state)
4: Choose an action A_t using policy derived from Q (e.g. ϵ-greedy)
5: Execute action A_t
6: Observe next state S_{t+1}
7: Collect immediate reward R_{t+1}
8: Update Q-value as follows: $Q(S_t, A_t) \leftarrow (1-\alpha)Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_{a \in A} Q(S_{t+1}, a)]$
9: Set $S_t \leftarrow S_{t+1}$
10: Optimal policy: $\pi^*(s) = \text{argmax}_{a \in A} Q(s,a)$

Figure 2.3. Q-learning algorithm.

$$Q_{target}(S_t, A_t) = R_{t+1} + \gamma max_{a \in A} Q(S_{t+1}, a) \tag{2.6}$$

Having obtained a target in line with SARSA, Q-learning or some other technique with respect to a certain state-action pair (S_t, A_t), we can finally use it to update Q by executing a rule similar to the one shown below. Where the extra parameter α is known as the learning rate and the difference between a the target and the current prediction is called the TD error.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(Q_{target}(S_t, A_t) - Q(S_t, A_t)) \tag{2.7}$$

Figures 2.2 and 2.3 present pseudo-codes corresponding to the entire operations of SARSA and Q-learning.

2.3.2 Policy Gradient (PG)

PG applies an opposite approach to TD methods in regard to learning. While the latter learns value functions that implicitly define policies, the former entirely disregards value functions and directly learns in the policy space. Hence, the goal of PG is to learn a parametrized policy π_θ that given a set of free parameters θ receives states as inputs and outputs a probability distribution over actions in the general case.

To this end, first, a score function $\mathcal{J}(\theta)$ must be defined, that indicates how good policy π_θ really is. Abiding by the optimality criteria established earlier, this score function is set in terms of the cumulative reward the agent would gather on average per episode starting from the initial state S_0:

$$\mathcal{J}(\theta) = E_{\pi_\theta}\Big[\sum_{k=0} \gamma^k R_{k+1} | S_0\Big] = v_{\pi_\theta}(S_0) \tag{2.8}$$

Afterwards, PG alternates between two steps: policy interaction and policy improvement. In policy interaction, the learning agent enforces the current policy π_θ in its environment as it gathers examples of the form $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$. Then, in policy improvement, gradient ascent is employed to maximize the score function with respect to its parameters θ. This is done by using the collected examples of every complete episode to compute the direction of improvement $\nabla \mathcal{J}(\theta)$, and lastly by applying the following update rule:

1: Initialize policy parameters θ
2: Repeat:
3: Generate an episode $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$ using $\pi_\theta(A_t|S_t)$
4: Compute gradient from collected data $\nabla_\theta J(\theta)$
5: Update parameters $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Figure 2.4. Policy gradient algorithm.

$$\theta \leftarrow \theta + \alpha \nabla_\theta \mathcal{J}(\theta) \tag{2.9}$$

Figure 2.4 provides a rough outline of a PG implementation. An important advantage of PG over TD is that it enables the application of reinforcement learning in very large or even continuous action spaces, since it learns the policy smoothly, instead of having the compute it at every step by maximizing over the entire action space associated with each state. A disadvantage of PG is that it is episodic in nature, i.e. it needs information of whole episodes in order to determine $\mathcal{J}(\theta)$ and then $\nabla \mathcal{J}(\theta)$; this can lead to a slow convergence in some domains.

2.3.3 Actor Critic (AC)

AC is a hybrid that combines temporal-difference and policy gradient approaches. It considers two separate learning structures (see Figure 2.5); on the one hand, there is a critic responsible for learning a parametrized action-value function; and on the other, there is an actor that directly learns a parametrized policy and that is in charge of selecting the agent's actions. The improvement of both models toward more accurate value predictions and more rewarding actions, respectively, is single-handedly driven by the TD error signal connected with the critic.

Thanks to their construction, actor-critic methods take the best of both worlds. They can handle very large or continuous action spaces just like PG, because they likewise train an explicit policy; while they also take advantage of the fast convergence of TD, since they also apply bootstrapping, which allows them to potentially learn after each interaction step.

2.4 Deep Reinforcement Learning

Deep reinforcement learning (DRL) combines classical RL with deep neural networks. The main benefits of this fusion are: 1) DRL better handles very large or continuous state and action spaces, 2) DRL is able to somewhat generalize across states, i.e. to provide optimal actions in states never encountered before, a feature lacking in classical tabular RL.

This section will start by listing some terminology relative to neural networks, and afterwards hallmark DRL algorithms will be presented.

2.4.1 Neural Networks Terminology

Below, it is presented a list of numerous terms commonly appearing in any work involving artificial neural networks, as it is the case of this document.

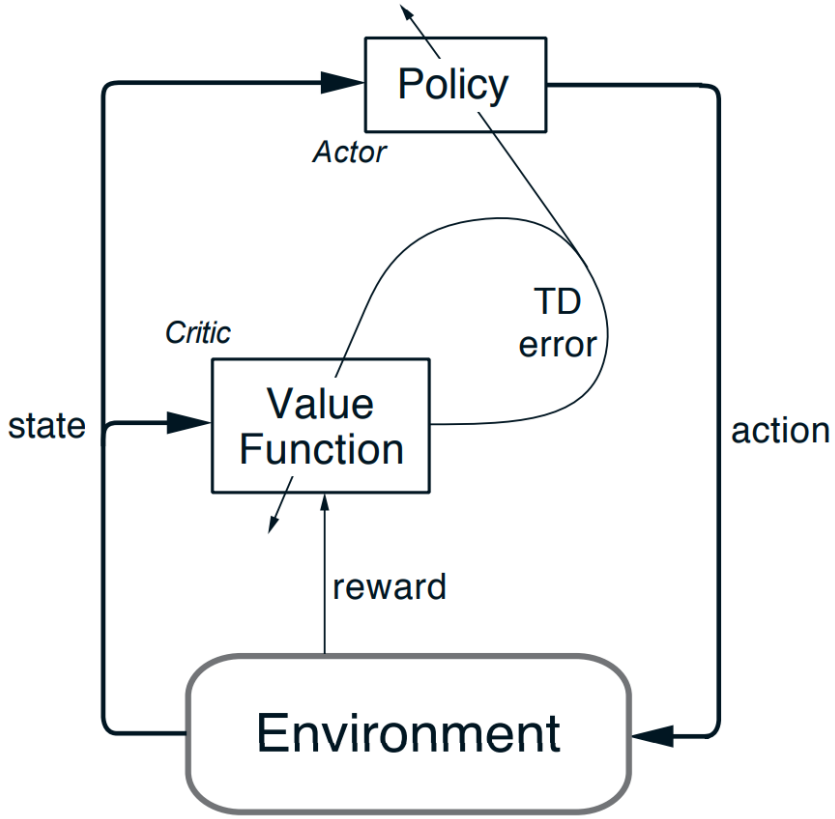


Figure 2.5. Actor-critic architecture.

- Neuron: Also called unit. It is an element that performs a basic mathematical operation. Normally, it receives inputs and sometimes a bias in the form of numbers, combines them by performing a weighted sum and finally passes the result through an activation function producing a numerical output.

- Neural Network: Any arrangement of several neurons working together (i.e., receiving the same input stream). For the purpose of machine learning, a neural network is simply a mathematical function with numerous degrees of freedom that help us approximate real-life functions. This is achieved by tweaking its internal variables over and over again until its output for a given input matches with good enough accuracy samples of the real-world function we wish to model.

- Input: Numerical data that is sent to a neural network. It could take many forms, such as: RGB images, real-valued vectors and more.

- Output: Numerical results that are generated by a neural network. It groups the individual results of multiple neurons and it could be structured in as many ways as in the case of inputs.

- Weights: Variable coefficients used by each neuron to perform a weighted sum of its input. Each neuron could connect with the outside world or with other

neurons through various weights.

- Bias: Singular variable unique to each neuron that shifts usually via addition its weighted sum between weights and input.
- Activation function: It is a plain mathematical function that is applied by neuron just before generating its output. Some examples include: linear, tanh, ReLU, leaky ReLU, ELU, softmax.
- Layer: It is a sub-grouping of neurons within a neural networks. These networks can be structured sequentially into numerous layers, where each receives as input the output of a preceding layer.
- Output layer: Layer that directly generates the output of a network. Usually, there is no other layer after it.
- Hidden layer: Any layer that is not an output layer.
- Dense layer: Also known as Feedforward layer. A layer whose neurons do not have connections through feedback loops. That is, they do not connect to themselves or to other neurons in the same layer or to neurons in layers preceding theirs.
- Recurrent layer: A layer that unlike a dense one has neurons presenting feedback loops. Two commonly-used recurrent formulations of a neuron are: LSTM and GRU.
- Convolutional layer: It is another type of non-recurrent layer, which instead of performing a full matrix multiplication between its input and weights as done by a dense layer, computes a convolution. As a gross oversimplification, in a convolution, we slide a small neural window over a tensor and process patches of its data, one by one. Later, these partial results are reorganized in a way that tries to preserve spatial correlations.
- Recurrent network: A neural network having at least one recurrent layer.
- Convolutional network: A neural network having at least one convolutional layer.
- Shallow network: A neural network having up to 2 hidden layers.
- Deep network: A neural network having any number of layer higher than 2.
- Example: Sample taken from the real-world function we would like to approximate. It must have two components: what goes in and what goes out of said function.
- Dataset: Corpus of examples.
- Targets: One of the two components of an example, corresponding to what is produced by a sampled function.

- Loss function: Also known a cost function. It is a mathematical relationship that we must establish between the output of a neural network and the targets within a dataset. Usually, the result of this operation is a singular real number. This relationship must be special in that if said real number becomes smaller this should generally translate into our output being more similar to the targets. An example is the mean square error.
- Optimizer: It is the algorithm responsible for tweaking (i.e., training) the variable weights and biases (also called parameters) of a neural network. Ideally, it would train them in a way that makes the loss function smaller. Currently, most optimizers used in deep learning a based on Stochastic Gradient Descent (SGD). A couple of examples are ADAM, Adagrad, Adadelta and RMSProp.
- Accuracy: Like the loss function, it is another mathematical measure of similarity between the output of a neural network and the targets of a dataset. However, it has less restrictions in its formulation and it is not employed anywhere during the optimization procedure. Typically, it is evaluated from examples that have not been seen by the optimizer.
- Udpate: Each individual training procedure, typically involving a significant amount of examples and possibly multiple individual optimizations. In reinforcement learning, we normally want to do multiple updates because we do not have all data available at the start of a session.
- Learning rate: A real number hyperparameter, which within the scope of SGD defines how big of a step we will take in the parameter space (i.e., weights and biases) in the direction of the antigradient. Essentially, it limits how much we can change the current parameters of the network in each update.
- Batch: Set of examples taken into account in a single update.
- Epoch: Number of consecutive optimizations perform over the same batch.
- Training session: Also called a run. An sequence of updates, usually ended when our neural models have reached a desired accuracy.

2.4.2 Deep Q-Networks

DQNs [65] are essentially the implementation of classical Q-Learning (although other TD methods can be implemented in the same way), while using deep neural networks to approximate the action-value function to be learned. Different deep neural architectures have been proposed for representing Q-functions (see Figure 2.6). For example, three common structures are: 1) using a single network fed with a state and an action, which outputs a single action value pertaining to the input pair, 2) using a single network fed only with a state, which outputs the action values for every action $a \in A$, 3) using multiple networks, as many as the cardinality of the action set $|A|$, each fed only with a state, so that each would output the action value corresponding to a different action.

When working with DQNs, the prediction step of standard Q-Learning is replaced with the tuning of the deep neural network with parameters θ. This training

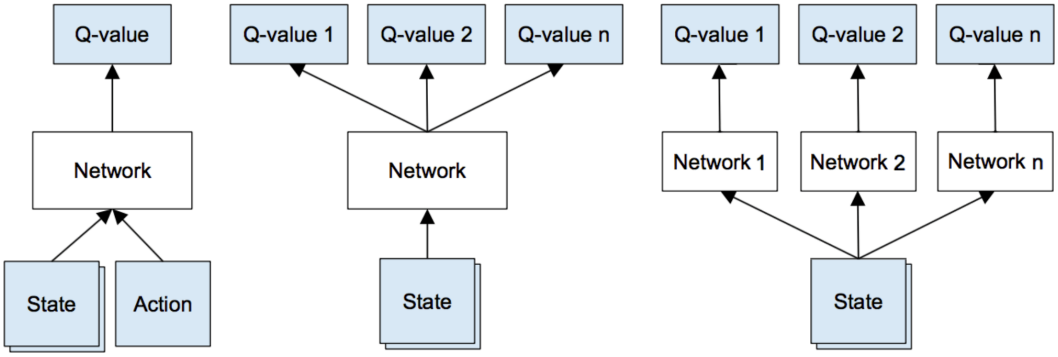


Figure 2.6. Policy gradient algorithm.

is performed at the level of each state-action pair. And the targets provided to the network are computed exactly as in standard Q-Learning. For instance, considering the first architecture shown in Figure 2.6 and the collected experience tuple $\langle S_t, A_t, R_{t+1}, S_{t+1} \rangle$, the target associated with pair (S_t, A_t) will be $R_{t+1} + \gamma max_a Q(S_{t+1}, a|\theta), \forall a \in A$. Where the second component of the previous sum indicates that we need to query the Q-network as many times as actions are in set A, always pairing an action with the next state that has been observed, and at last taking the maximum Q-value among all these outputs.

Moreover, in DQN, the improvement step is implemented only partially, i.e., the policy is never expressed in its full explicit form. Instead, whenever the agent, as it interacts with the world, has to select an action according to its underlying policy, what is done is that the current state is fed to the DQN, which computes the Q-values associated with every possible action, and finally the action with the highest Q-value is greedily chosen as usual.

If DQN was to be implemented as described above, learning would likely turn out to be unstable. From a practical standpoint, two extra mechanism are necessary to avoid such an inconvenience. One cause for this instability comes from the fact that when performing SGD-grounded optimization over deep networks it is assumed that samples are independent and identically distributed. However, that is not the case in DRL, examples collected from the same episodes are correlated to each other, and over usually larger scales, examples are biased by the preferences of the current policy. Thus, the first mechanism is an experience replay buffer. This is a large memory with sufficient capacity to store experience tuples spanning across numerous episodes and network updates. With this element in place, before each update of its Q-network, DQN draws a batch of random experiences from this replay buffer. A selection process that effectively removes correlations existing in the sequentially observed examples.

When using a neural network to approximate a Q-function, each update aimed at improving a single action-value will necessarily modify several weights of the network, and consequently, this will also modify in one way or another all current predictions of the network. Since both, DQN relies on bootstrapping, the fact that each update alters the targets used to compute that very update is a major source of instability during learning, usually causing oscillations or divergence of the policy. For this reason, a second essential mechanism that will help with this problem is a

target network. In the context of DQN, such a target network is a separate deep Q-Network with its own parameters θ'. Yet, it has the same functionality, that is, it is also trained to predict action-values. However, a key difference is that this additional network is updated much less frequently; once every several updates of the original one. As a result, this target network is the one used to compute targets as it provides slow-varying value estimates that prevent destabilizing the learning process.

A common way to smoothly update a target network is to take a large moving average with respect to the original model given a small learning rate τ, resulting in:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \tag{2.10}$$

Due to the addition of a target network, the update rule of DQN is modified roughly as follows:

$$\mathcal{L}(\theta) = (R_{t+1} + \gamma max_a Q(S_{t+1}, a|\theta') - Q(S_t, A_t|\theta))^2 \tag{2.11}$$

$$\theta \leftarrow \theta - \alpha\nabla_\theta \mathcal{L}(\theta) \tag{2.12}$$

2.4.3 Deep Deterministic Policy Gradients

DDPG is a straightforward DRL extension of the actor-critic architecture, consisting of two separate deep neural networks. The critic learns a parametrized Q-function $Q(s, a|\theta^Q)$ Meanwhile, the actor learns and executes a parametrized policy $\pi(s|\theta^\pi)$. Having an explicit model for its policy makes DDPG applicable in problems with continuous actions spaces.

The critic is trained in a similar fashion as a Q-network in DQN, by applying gradient descent to minimize the error between predictions and targets estimated thanks to bootstrapping. However, the targets are no longer estimated from a maximization procedure, since that becomes very expensive in continuous action spaces. Now, the best action in the next state is guessed to be the one the current actor selects, $A_{t+1} = \pi(S_{t+1}|\theta^\pi)$. And it is expected that this assumption gradually becomes true as the actor network converges. Thus, targets will be computed as: $R_{t+1} + \gamma Q(S_{t+1}, \pi(S_{t+1}|\theta^\pi)|\theta^Q)$.

On the other hand, the actor's goal is to minimize for each example the difference between its current output and the optimal action. In theory, the optimal action could be estimated from the critic, after realizing a maximization over the entire action space given a particular state. Yet, this is too expensive. Hence, instead, the actor is trained in a smarter way by taking advantage of the gradients of the critic with respect to its action inputs, which are computed only after the critic has been updated with the latest targets. To complete the chain rule, such a gradient term is then concatenated with gradient of actor with respect to its own parameters, resulting in the following vector of improvement for θ^π that internalizes recent improvements in Q-values: $\nabla_a Q(S_t, A_t|\theta^Q)\nabla_{\theta^\pi}\pi(S_t|\theta^\pi)$

Without a proper implementation, DDPG would suffer from the very instability that affects DQN. Because of that, this algorithm adopts the same two mechanisms introduced by the latter method. This means that DDPG also maintains a sufficiently

large experience replay buffer from which it samples past examples as needed. Furthermore, this methods keeps two target networks, one relative to the critic and the other to the actor, which are trained and operate exactly as explain previously. If such target networks have parameters θ'^{Q} and θ'^{π}, the final update rules for the original networks are:

$$\mathcal{L}(\theta^Q) = (R_{t+1} + \gamma Q(S_{t+1}, \pi(S_{t+1}|\theta'^{\pi})|\theta'^{Q}) - Q(S_t, A_t|\theta^Q))^2 \tag{2.13}$$

$$\theta^Q \leftarrow \theta^Q - \alpha^Q \nabla_{\theta^Q} \mathcal{L}(\theta^Q) \tag{2.14}$$

$$\theta^\pi \leftarrow \theta^\pi - \alpha^\pi \nabla_a Q(S_t, A_t|\theta^Q) \nabla_{\theta^\pi} \pi(S_t|\theta^\pi) \tag{2.15}$$

2.5 Exploration Strategies

A famous maxim in reinforcement learning says: to learn an optimal policy one must try every state action pair infinitely often. This is a direct consequence of the fact that, as detailed in previous sections, all reinforcement learning algorithms use samples to approximate distributions. And as with any statistical learning approach, to reconstruct such distributions as correctly as possible, those samples have to be independently and identically distributed. A more down to earth intuition is that, in any task, we should have some knowledge of every part of the world before truly being able to decide which is the best strategy.

Taking each and every single time the best action according to the information we have received so far is known as an exploitation strategy. By definition, this action selection rule does not abide by the previous maxim, and as we will see in the next example it produces poor results most of the time. Imagine a world with a finite and sufficiently small number of states where all rewards are positive. There, we can choose to apply Q-learning and keep a tabular representation of the Q-function, where every value is initialized to zero. Let's say now that at some instant the learning agent lands in state S and there it can execute one of two actions A or B. Furthermore, regardless of which it executes, it gets some reward. Obviously, the first time it reaches S the Q-values for both actions will be zero; thus, to make a decision the algorithm needs a tie-breaking rule. For example, alphabetical ordering. In that case, action A will be arbitrarily chosen and from then on its value will forever remain positive (because all rewards are positive). At this point, it is easy to see that if our agent exclusively enforces a exploitative strategy in state S, it will always select action A (positive value) and never action B (zero value). Therefore, it will blindly follow this strategy without ever knowing the true action-value of B. Such a value could have very well been higher than that of action A, in which case, our naive agent has terribly failed to learn an optimal policy.

This is why in practice, every DLR method alternates between doing exploitation and choosing action in a different way, but that complies with the initial maxim. That is, it has to select each possible action in a given state every now and then. Whichever control mechanism that does this is called an exploration strategy. Some of the most ubiquitous exploration strategies are briefly explained next:

- ϵ-greedy: This rule defines beforehand a threshold, denoted ϵ; a real number usually between 0 and 1. Then, each time an action must be chosen, it draws a number at random from within the former range and applies the following condition. If the number is lower than ϵ, the learning agent selects an action at random from set A; otherwise it enacts its estimated optimal action. This technique was the one employed in the highly-acclaimed implementation of DQN [9].

- Boltzmann exploration: Also called softmax or Gibbs exploration. This strategy is applicable purely when using learners that compute a Q-function. It assigns each action a sampling probability that depends on its current Q-value estimate, according to the following equation, where T is a temperature parameter:

$$\pi(a|s) = \frac{e^{Q(s,a)/T}}{\sum_{a_i \in A} e^{Q(s,a_i)/T}} \tag{2.16}$$

- Entropy regularization: This is more alike a scheme rather than an explicit rule or formulation, as were the other techniques within this list. It can only be enforced while using a policy gradient method and when we have stochastic policies, that it, policies that assign probabilities to each action. In this case, we add an extra loss term to the loss function used for adapting our policy, which already contains at least one other term trying to maximize rewards. This new loss tries to push the output probabilities of our stochastic policy toward having maximal entropy (i.e., being shaped as a uniform distribution).

- Action space perturbation: This is a rather straightforward formulation used when we have continuous action spaces. Several types of noises have been proposed to serve as perturbations. However, the simplest approach to create one is to draw as many random numbers as dimensions our action space has and just add them to the optimal action. DDPG relies on this kind of exploration, though it adopts an Ornstein-Uhlenbeck process as a source of noise.

- Noisy networks [30]: This strategy is analogous to the previous one, but here we add a perturbation to the parameter vector corresponding to the model from which we extract actions. For example, this vector could represent the weights and biases of the neural network playing the role of our policy or of our value function approximator.

Chapter 3

Reward Sparsity

This chapter seeks to paint a clearer picture of why the lack of reinforcement (i.e., rewards), the obvious cornerstone of reinforcement learning, is so troublesome for this learning framework. A beyond that, it also compiles a number of ideas that have been specifically proposed as a way to circumvent this issue.

3.1 The Issue with Sparse Rewards

We say that an environment has sparse rewards, when the latter are zero almost everywhere, and the few positive incentives that might exist are far from each other and from every initial state. So far in fact, that a uniform policy (i.e., every action has an equal probability of being selected) cannot make its way from one rewarding state to the next purely by chance in any practical time.

It is easily seen why this is problematic. Recall from Section 2.2 that the core objective of reinforcement learning is to find a policy that maximizes some notion of return. In sparse-reward domains, by behaving uniformly, every reward we see is usually zero; meaning that, when computing returns, we will get zero every time. In other words, the landscape of the return function that we are able to approximate with the available information is flat or nearly flat. And at this point, it is evident that we have already run into a major problem. Mathematically, there is no meaningful way to maximize a flat function; in more technical terms, it is an ill-defined optimization problem. From this it follows that, it is impossible to find an optimal policy and thus, the reinforcement learning approach fundamentally fails when rewards are sparse. Numerically, the most sensible resolution to this indefiniteness is to assign equal importance to every point in space and to consider every action as equally good. And this is equivalent to learning an optimal policy that takes actions evenly. But that put us in an endless loop of disappointment, as by definition, such a strategy is extremely unlikely to discover any reward, which would be the only way out of this entrapment.

Does this analysis conclude that reward sparsity is mathematically intractable and there is no solution for it? Not quite. The previous reasoning takes three critical assumptions that are generally true in the literature. In particular, we have unsoundly presumed: that a learner behaves uniformly from the start, 2) that it will always act according to its optimal policy and 3) that only reward is capable of

promoting intelligent behavior. But the reality is that an algorithm does not need to abide by any of these premises. It could perfectly well employ exploration strategies that have inherent biases (e.g., in a large corridor moving always to the left would be effective 50% of the time). And it could as well define progress in terms of other environmental signals; for instance, it could be self-motivated to navigate toward observations it has not seen before (i.e, novelty). And as a matter of fact, solving sparse-reward environments with reinforcement learning might only be possible by precisely derailing from such widely accepted assumptions. Indeed, many of the most famous general DRL methods of recent years embrace those presumptions, and consequently, they fail to handle reward sparsity. Across the board, they strictly have the lone goal of maximizing reward; plus, they rely on the very simple exploration strategies described in Section 2.5. All of which select actions evenly beyond the time frame of a few tens of training updates of the neural models involved in learning (e.g., policies or value functions).

3.2 Approaches to Reward Sparsity

The inherent difficulty of sparse rewards demonstrated by the poor results recounted in the previous section impressed DRL researchers so profoundly that over the years they have developed numerous end-to-end algorithms that expressly address this issue. Along this endeavor, a wide range of disparate avenues for dealing with reward sparsity have been explored, often by breaking off from the assumptions described in the former section, and this led to the production of several impressive solutions. The coming subsections briefly explain some of the most fundamental ideas wielded in our battle against sparse rewards.

3.2.1 Reward Bonuses

Bonus-based exploration reformulates the reward function, also denoted as task or extrinsic reward, of the original MDP by adding a bonus to it, sometimes called intrinsic reward. Since we wish to sidestep sparse settings, we should prefer dense reward bonuses, i.e. bonuses that are non-zero for almost every state. Moreover, given that their ultimate motivation is to find all existing task rewards, it is commonplace to conceive bonuses that promote structured exploration of the environment; as it is done by the following methodologies:

State novelty has been used as a reward bonus in tabular settings for a long time. There, the learner is granted a novelty bonus every time it enters a state and the novelty of each state is inversely linked to its visitation count, i.e. the number of times the agent has visited that state since learning began. Current DRL solutions intended for very large or continuous state spaces [9, 106, 73, 64, 31] typically approximate this count, differing mainly in the details of their computations. Critically, visitation count always increases, which means that novelty will approach zero as time goes by and the original reward function will be eventually recovered

Intra-life state novelty is first investigated in [98]. The main idea is to reset all visitation estimates back to zero at the end of every episode, while computing novelty as usual during a single episode. That study in particular employs a first-visit count

such that novelty is zero for any subsequent visit to the same state. Conversely, novelty associated with life-long visitation counts is dubbed across-training novelty.

State curiosity seeks to learn a predictive model that takes the current state and possibly other variables as inputs. Several models has been suggested, such as: forward dynamics, inverse dynamics, random network embeddings among others. Based on this model, curiosity is then defined as a measure of how far is the prediction from the ground truth, e.g. prediction error, at each state [97, 74, 11, 39]. Therefore, curiosity bonuses will drive the learner to explore less-understood regions of its world. Similar to novelty, curiosity will inevitably wither down over time.

We have presented here an extremely short list of reward bonuses. Thus, for completeness, we would like to mention a few other compelling ideas: learning progress [85], information-gain [44], compressiveness [49], value curiosity [94], empowerment [66], disentanglement of bonuses [10].

3.2.2 Bayesian Uncertainty

Another compelling line of research uses approximations to Bayesian uncertainty as guidance for deep exploration. The key idea here is to focus exploration on areas of the world where our predictions still report a significant variance, and specifically, a large variance originating from a lack of samples. In [72], exploration is enforced by training a multi-head Q-network and, in the spirit of Thompson sampling, by randomly selecting a head at the beginning of each episode and following its induced policy thereafter. In [69], an extension of the Bellman equation is formulated, thanks to which it is then possible to learn the means and variances of Q-values. With the help of a Q-network trained to concurrently estimate both statistics, this method finally employs a selection strategy designed to prefer actions leading to higher returns with higher uncertainties. In [90], Bayesian uncertainty is approximated as the ensemble disagreement from 5 models trained in parallel to predict one-step transitions. Afterwards, these very uncertainties are taken as intrinsic rewards and sent to a base learner for their maximization, a procedure that generates some intelligent behavior despite task rewards being hard to come by.

3.2.3 Skill Discovery

A skill is a mapping from states to actions or other skills that is meaningful within a limited region of the state space; namely, a local policy. An option is a skill represented by a triple containing an initiation set, a policy and a termination set [104]. One way to deal with sparse rewards is to learn a policy directly over skills or options instead of primitive actions; which improves exploration as the agent can reach further with fewer decisions. If our goal is end-to-end DRL, then we must learn these skills automatically; and for this we might rely on:

Skill diversity concurrently trains a set of policies on the target problem while ignoring task rewards. Usually, all policies are encoded within a single skill-conditioned policy parametrized by some latent variable. Policies are optimized independently based on several objectives that may differ between specific algorithms [34, 23, 1, 91], but a common one is that each policy must be distinguishable from every other. The end result is a diverse set of skills.

Option hierarchy encompass solutions that simultaneously learn options and a hierarchical policy over those options [4, 40, 82, 95, 120], i.e. a mapping from states to options. Options and hierarchical policy are trained together by reusing experiences across all options and propagating extrinsic rewards throughout the underlying architecture.

Goal hierarchy likewise promotes a hierarchical framework, which in this case learns a goal-conditioned policy and a hierarchical policy over goals at the same time and from a single stream of observations [111, 67, 59]. Critically, the hierarchical policy is trained solely on task rewards, while the goal-conditioned policy executes unsupervised mastery.

3.2.4 Automatic Curriculum

This approach modifies the specification of the original RL problem in a controlled fashion, in an effort to automatically generate a sequence of tasks of varying difficulty that are still related to each other. Afterwards, these tasks are presented to the learning agent in a suitable order, in the hope that as the learner overcomes the easier problems it will build up competence for solving the remaining ones. There are several ways to implement a curriculum, some of which are reported below:

Mastery seeks to learn a goal-conditioned policy in an unsupervised manner, i.e. without extrinsic rewards [28, 109, 68, 110, 75, 24]. Said policy would bring the agent to any valid goal state, oftentimes from any initial state. To this end, this approach maintains a goal set from which a goal is drawn at the beginning of every training episode. Crucially, the goal set is empty when learning starts, i.e. the learner is initially not aware of which goals exist. This set is then augmented as new states (encoding new goals) are visited. Mastery-driven methods hope that as the agent becomes more competent given the current set of goals, it will more likely arrive at states that were too far away before, adding their goals to the ongoing set; this cycle will repeat until the entire goal space has been accounted for.

Reverse curriculum alters the set of initial states (S0) of the target problem [29]; such that when learning starts, S0 is very close to or includes at least one goal state, and as learning progresses and the agent explores more of its world, S0 is continuosly updated with new states that are further away from the reference goal state. As a result, the learner will gradually solve harder tasks with time.

Self-play curriculum considers two learning agents competing against each other as they take turns interacting with a common environment [100, 99, 61]. The first agent proposes self-play tasks to the second one by performing rollouts that the later must imitate, either by reaching the same final state within a similar amount of time or by following the exact same trajectory. While the second learner gets rewarded for an accurate impersonation, the first one is positively reinforced for proposing ever more diverse and challenging self-play tasks. Agents involved in a self-play curriculum are expected to explore their environment and gain an understanding of how it works even in the absence of extrinsic rewards.

3.2.5 Hindsight Learning

Techniques of this kind relabel the goal originally commanded to the agent with some other goal actually achieved by the same learner while trying to complete the task [2, 80, 81]. In other words, if an agent was asked to do A but it does B, this is still informed as a success to the optimization module by declaring that B had been requested all along and by adjusting accordingly the terminal reward granted to the agent. This results in accelerating learning for goals known to be achievable until the current iteration.

3.2.6 Actionable Distances

Algorithms in this category aim at learning, in an unsupervised fashion, a distance function between any two states that internalizes the dynamics of the target environment. Ideally, this function would encode the minimum number of primitive actions needed to go from one state to another. Afterwards, during a second learning phase, such distance function is used to compute shaped rewards that will guide a standard DRL agent towards any goal specified in the same domain. Some notable works directly learns a distance function [27] and others learn actionable embeddings [33, 116], such that distances between embeddings represent actionable distances.

3.3 Benchmarked Algorithms

Here, three successful algorithms capable of dealing with sparse rewards to some extent are briefly introduced. These will be heavily utilized between Chapters 5 to 9.

3.3.1 Random Network Distillation (RND)

RND [12] marked a performance milestone in Montezuma's revenge. And even today, its achieved score stands as one of the highest for an end-to-end DRL solution. In RND, exploration is driven by novelty, that is, this learner rewards itself for seeing new or rare observations. Specifically, RND keeps a fixed randomly-initialized network that generates embeddings from observations and trains a second network to predict those embeddings. Afterwards, intrinsic rewards are computed as the Euclidean distance between random and predicted embeddings; thus, they are smaller for observations perceived more often.

3.3.2 Intrinsic Curiosity Module (ICM)

ICM [74] is another greatly influential approach. Insomuch that it has been assimilated by NGU and Agent 57 to produce an impressive general-purpose algorithms. The behavior of ICM is grounded on curiosity. In particular, this method relies on the prediction error stemming from a trained one-step forward dynamics model to quantify curiosity, which is then used as an intrinsic reward that guides exploration towards not yet well-predicted transitions. Critically, ICM also trains an inverse dynamics model that shares a representation with the forward one, and this has the benefit of keeping out from the computation of curiosity elements unaffected by the actions of the interacting agent.

3.3.3 Episodic Curiosity Through Reachability (EC)

EC [84] made a breakthrough in the field by solving the noisy TV problem. In a nutshell, an EC agent awards itself a high intrinsic reward every time it visits a hard-to-reach state. By continuously adding such states to an episodic memory, any future state is labeled as hard-to-reach only if it requires at least k actions to get to from any state already in the memory. To compute reachability, EC trains a neural classifier from pairs of states that were less or more than k-steps apart in past trajectories. In the standard version of EC, this classifier is pretrained thanks to a random policy; while in the online version, called ECO by its authors, the reachability network is learned concurrently with the agent's policy. By construction, EC is expected to ignore any distractor that demands a low transitioning effort; such as: pixel-level noise or a broken/noisy TV.

Chapter 4

An Overview of Environmental Features that Impact Deep Reinforcement Learning in Sparse-Reward Domains

Deep reinforcement learning has achieved impressive results in recent years; yet, it is still severely troubled by environments showcasing sparse rewards. On top of that, not all sparse-reward environments are created equal, i.e., they can differ in the presence or absence of various features, with many of them having a great impact on learning. In light of this, the present chapter puts together a literature compilation of such environmental features, covering particularly those that have been taken advantage of and those that continue to pose a challenge. This effort is expected to provide guidance to researchers for assessing the generality of their new proposals and to call their attention to issues that remain unresolved when dealing with sparse rewards.

4.1 In Pursuit of Domain-Generality

Notwithstanding the extraordinary achievements made over the years by numerous algorithms purposefully constructed to work with sparse rewards, one persistent issue that affects most of them is their lack of generality. Which is to say, their applicability and effectiveness depends heavily on the specific features of the environment being tackled. This fact has already been pointed out by Taiga et al. [105], though phrased slightly differently: good performance in a given sparse-reward environment is not indicative of good performance in other such scenarios. For instance, RND [12], despite being the first end-to-end DRL solution to reach the current state-of-the-art score in Montezuma's revenge, performs poorly in environments that contain distractors or that demand it to retrace its steps. Meanwhile, other algorithms have specialized on exploiting particular environmental features. To give an example, reverse curriculum generation [29] is a rather ingenious solution applicable only in environments that are resettable and reversible.

4.2 Contributions of this Overview

To date, the knowledge of these environmental features that have a manifest impact on the performance of deep reinforcement learning algorithms, especially in sparse-reward environments, is found dispersed among independent studies. Thus, this work brings a contribution by composing an overview of such features, with emphasis on those whose presence facilitates learning as well as those that make it more challenging. Importantly, this study focuses exclusively on end-to-end single-agent reinforcement learning methods. Therefore, other promising techniques (e.g., imitation learning, advice based learning, multi-agent reinforcement learning) and algorithms (e.g., [5, 21]) will mostly not be reviewed. Apart from providing a deeper understanding of the intricacies of applying reinforcement learning in domains with sparse rewards, this work is further expected to offer guidance to researchers for assessing the generality of their newly proposed solutions. To this end, this overview additionally provides detailed descriptions of how (i.e., what mechanisms) and why (i.e., which components of an algorithm confer a weakness) these features affect learning. Then, equipped with such information, researchers could validate, from a theoretical standpoint, if analogous mechanisms unfold within their approaches or if they exhibit the same weaknesses. One last virtue of this overview is that it pinpoints several open areas of research, which correspond to features whose handling is still out of grasp for existing algorithms.

4.3 Impactful Environmental Features

The following subsections elaborate about a number of environmental features that according to a review of the literature have noticeable effects on the performance of reinforcement learning algorithms developed for dealing with sparse rewards. In each case, it is provided a definition of the given feature, useful examples and a synopsis of previous works that have originally looked into such features (all of which have already been outlined in Subsection 3.2). Particularly, every study included in the next sub-sections makes one of the following contributions. Exploits the occurrence of a feature to facilitate exploration in sparse-reward problems. Exposes how an algorithm otherwise successful at handling a lack of external reinforcement fails when a certain feature is present. Or, delivers a description, expressed as an hypothesis or supported by empirical evidence, of the mechanisms by which a feature impacts the machinery devised to conquer reward sparsity. Table 4.1 gives a preview of all the features that have been compiled in this study, indicating also which previous works have discussed, applied, internalized or experimented with them.

4.3.1 Resettability

A resettable environment lets researchers configure any valid state as the initial state of a given episode. For example, robotics simulators readily position a robot according to any desired joint angle and velocity configuration (see Figure 4.1). As this property explicitly allows to reformulate the original task, it has been exploited by several automatic curriculum approaches. All of them capable of exploring whole environments while receiving at most just one extrinsic reward at the end

Feature	Definition	Examples
Resettability	Designers have control over each episode's initial state	Simulated robotics environments are usually resettable
Reversibility	Being able to return to a previous state during an episode	An unclimbable cliff dividing the world into two strata is an irreversibility
Retracing steps	Having to re-visit past states during an episode	Searching for a key and then returning to a locked door under partial observability
Partially ordered subtasks	Having to perform diverse skills with any ordering being feasible	Multiple radially-expanding culs-de-sac
Distractors	Dynamic elements that are observable but irrelevant to the main task	Leaves moving in a breeze
Stochastic dynamics	Performing the same action in the same state leads to multiple outcomes	Sticky actions
Action-prediction degeneracy	Diverse actions lead to the exact same outcome	Pushing a rigid block into the ground
Deadly states	States that produce the early termination of an episode	Enemies, lava floors, falling from high elevations, etc.
Procedural generation	Each episode has a slightly different layout but the abstract goal remains the same	Varying geometries, textures, lighting, etc.

Table 4.1. Summary of environmental features contained in this study.

of a successful episode and many times none. For instance, Florensa et al. [29] enforce a reverse curriculum by altering the initial state distribution of a problem. In particular, this distribution at first is set to occupy a small volume around a terminal goal, which must be known beforehand. But afterwards, it is consented to gradually grow with every new state encountered, as this algorithm learns to reach said terminal state from further and further away. Ultimately, the ambition of this method is to engulf inside such a distribution the entire state space associated with a given task, while simultaneously developing a policy capable of solving the latter starting from anywhere in the world. To sum up, in practice, this method maintains in memory a set of previously seen states, which when training starts is manually populated with a few states located nearby the known goal. This technique then leverages resettability by forcing each episode to initiate in a state sampled uniformly at random from the previous set. Similarly, Eysenbach et al. [24] establish a curriculum that pushes the learning agent to master its environment locally. In practice, this algorithm trains a goal-conditioned policy to connect any two states that are close to each other. And to this end, in each episode it samples an initial state (from a anywhere in the world) plus a goal state (forced to be within 4 interaction steps from the start 80% of episodes), and then resets the environment to the former. In this way, during testing, if the given goal is determined to be near the

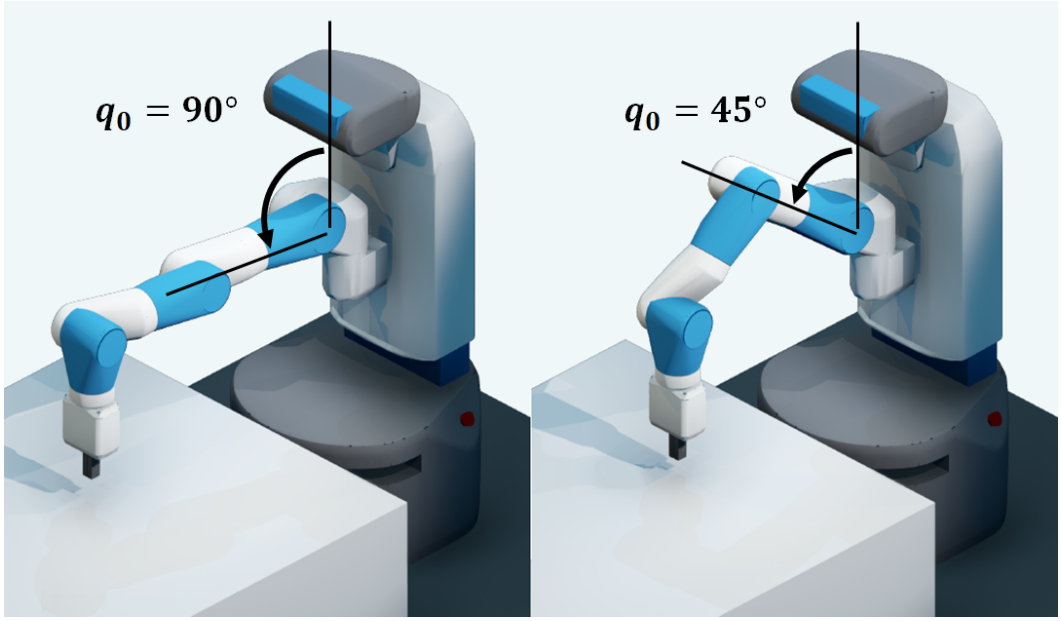


Figure 4.1. Simulated manipulator environments usually allow the resetting of their initial state to any desired joint configuration.

start, the learned policy is directly asked to reach such target. Otherwise, planning is applied to decompose the task by defining an ordered set of waypoints (i.e., states outlining the shortest path to the target). Of which, the first one is commanded to the policy at the beginning of the episode and the rest are issued one by one each time the immediate sub-goal is achieved. Figure 4.2 demonstrates how this method (called SoRB) takes advantage of resettability to outclass a powerful general-purpose approach such as C51 [7]. Sukhbaatar et al. [100] also relies on the aforementioned feature to generate a self-play curriculum (in one of their proposed variants at least). Such curriculum emerges from the interaction of two learning agents: a setter and an imitator. The setter proposes a task by executing a trajectory and it is rewarded when the task is too complicated for the imitator, which in turn is rewarded for getting to the same final state as the setter. It is thanks to this competition that these agents increasingly explore more of their environment. Critically, for this to work best, the imitator must be reset to the same or a very similar initial state as the setter, which ensures that the difficulty of self-play tasks is always the right amount.

4.3.2 Reversibility

In a reversible environment, if going from state A to state B is feasible for an agent, then moving from state B to state A is also feasible for any two states (see Figure 4.3). Moreover, in such domains, just like in any standard DRL problem, the agent has no a priori knowledge of the transition function and ergo of the action or the sequence of actions that would allow it to return to a previous state. Irreversibility is not an uncommon feature; it appears in many benchmark environments used to test reinforcement learning. For instance, it happens in videogames (e.g., when an avatar

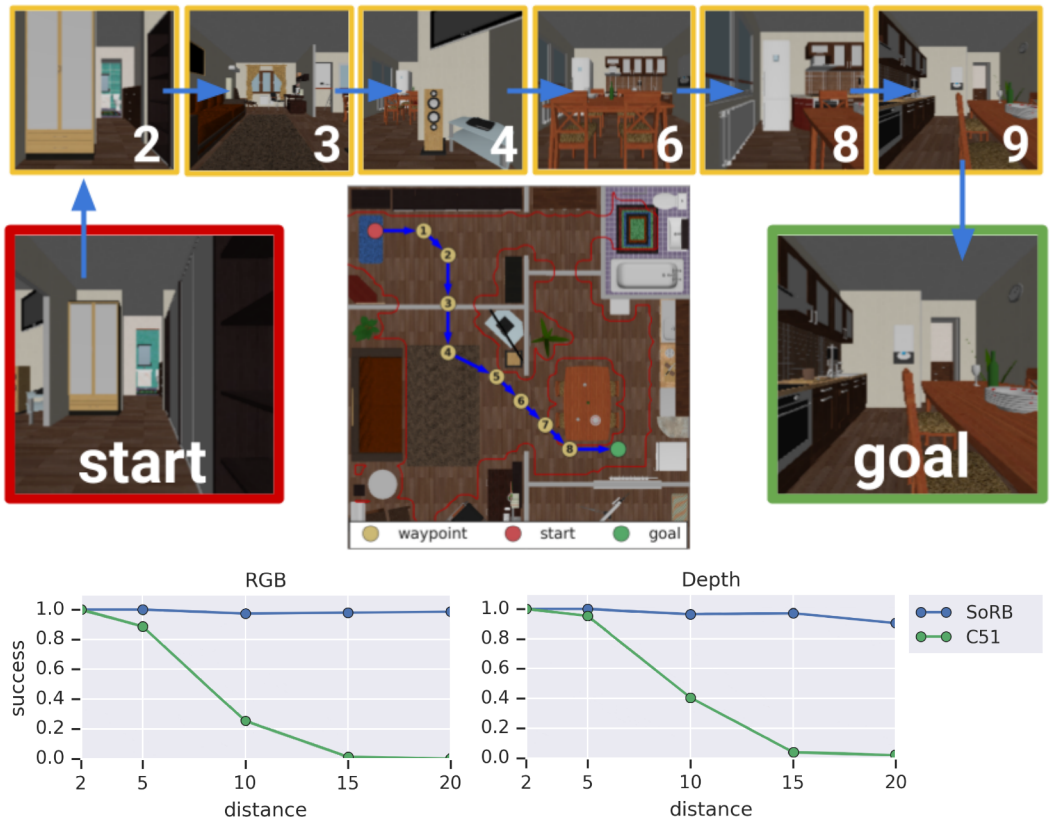


Figure 4.2. Comparison between SoRB and C51 in a sparse visual navigation domain (performed by Eysenbach et al. [24]). The task (top) requires an agent to traverse a virtual 3D house until arriving at a goal observation presented to it before the simulation initiates. A positive reward is only handed when the target is reached. The graphs at the bottom report the success ratio attained when observations are RGB images (left) and depth images (right). Their x-axes indicate distance between start and goal, measured in terms of interaction steps. These results clearly expose the superiority of SoRB, which as the distance to the target increases performs increasingly better than C51.

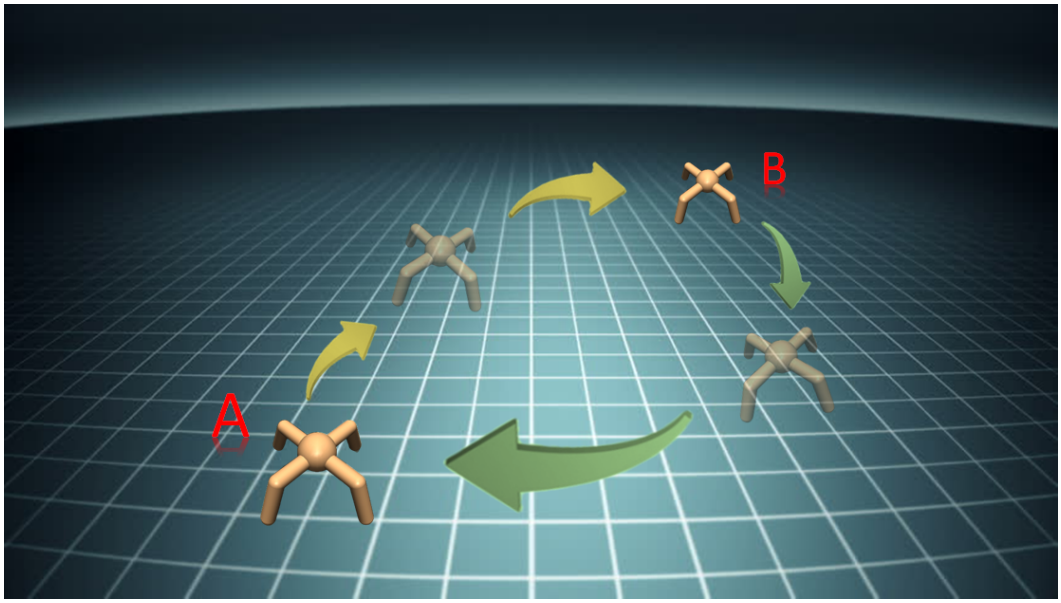


Figure 4.3. A flat and infinite arena is reversible since an agent that navigates from A to B sees no obstacle that forbids it to move in the opposite direction, from B to A.

falls off a cliff that it cannot climb back up or when a car agent rolls over because of an accident) and in robotics tasks (e.g., when a robotic manipulator throws an object outside its workspace). Reversibility is a necessary condition for the applicability of the proposal made by Florensa et al. [29], which when met along with resettability produces outstanding results (see Figure 4.5.) As reported in the previous subsection, this algorithm performs exploration backwards, starting from a known goal. However, if every trajectory leading to that target in the forward direction is irreversible (e.g., the goal is at the bottom of an unclimbable cliff, as depicted in Figure 4.4), then such reverse curriculum will never solve the problem in hand. Sukhbaatar et al. [100] too exploit reversibility in one of their self-play curriculum variants. As already described, this method creates a competition between a setter and an imitator. And in this variant, after the setter suggests a task, policy control is switched to the imitator, which is rewarded for moving from the current state to the setter's initial state (note that resettability is not required here). Hence, this approach is sensible only in reversible domains, since in the opposite case the setter would be compelled to always propose impossible tasks. Another work that puts reversibility in the spotlight is the one formulated by Grinsztajn et al. [35]. To be clear, this algorithm operates properly independently of whether or not an environment has irreversibilities. Notwithstanding, concurrently to a policy, it trains a neural model to predict if a transition between two input states is reversible or not using the temporal ordering found in past trajectories as ground truth. Said model is then used in up to two ways. To constrain exploration by avoiding irreversible actions. Plus, to guide exploration even in the presence of reward sparsity by allocating large internally-generated bonuses to transitions currently predicted to be reversible.

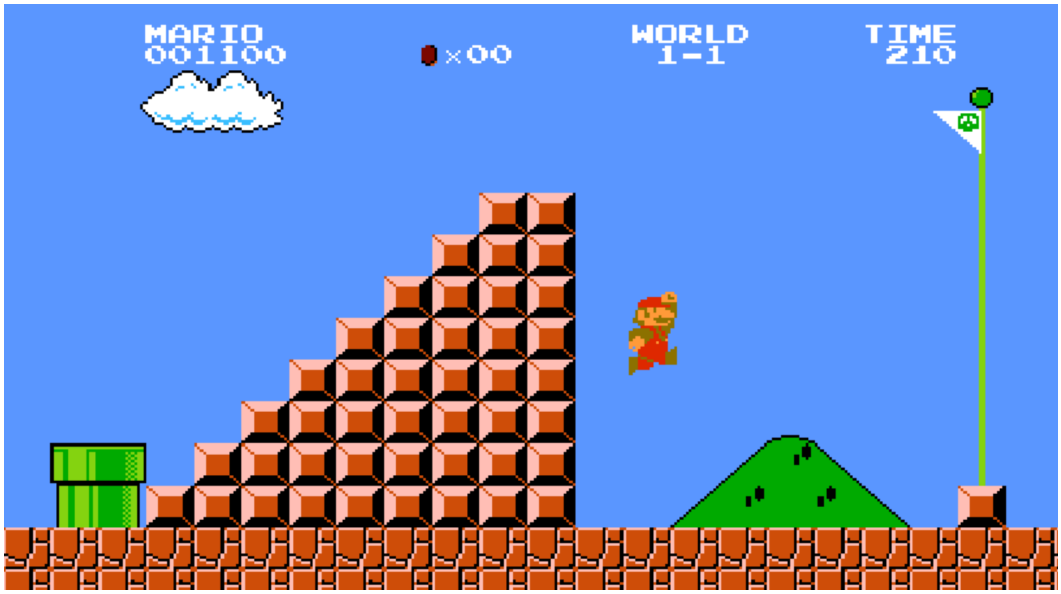


Figure 4.4. A irreversibility is present in the first level of the original Super Mario Bros, as Mario cannot jump back up the triangular structure located next to the goal pole.

4.3.3 Retracing Steps

A learning agent is forced to retrace its steps in any environment that demands re-visiting some states before reaching its goal. For an illustrative example, consider an agent that although being near a locked door, to open it, it must first search elsewhere for the corresponding key. Crucially, the agent has to navigate extensively through the world before finding said key, and then, to return to the locked door, it has no choice but to re-visit the same locations seen along the path leading to the key. In a successful episode, the agent gets one positive reward for opening the door and possibly another for collecting the key; no other state or transition gives any reward. Scenarios like the previous one are problematic for a wide range of algorithms designed to thoroughly explore sparse-reward domains. Specifically, the fact that two very similar (in fully observable environments) or even identical (in partially observable environments) observations correspond to two fundamentally different states (e.g., before and after collecting the key) can very well confuse the exploration machinery of said methods. They could wrongly assume that states belonging to the return trip have already been visited and that would result in them halting exploration prematurely. For instance, Roderick et al. [83] show that the issue of retracing steps severely impacts algorithms that employ across-training novelty-based bonus rewards. This work evaluates one of these approaches, namely DQN-CTS [9], in a version of Montezuma's revenge, modified such that the interacting agent is conceded just one life instead of five per episode. Note that the initial room of this game compels the agent to first find a key, then backtrack to its initial position and finally leave the room by unlocking one of two doors (as shown in Figure 4.6). Such experiment reveals that DQN-CTS cannot escape the first room of the game in the modified setup. The authors argue that this occurs because such learner is unable to disassociate an observation seen before collecting the key from

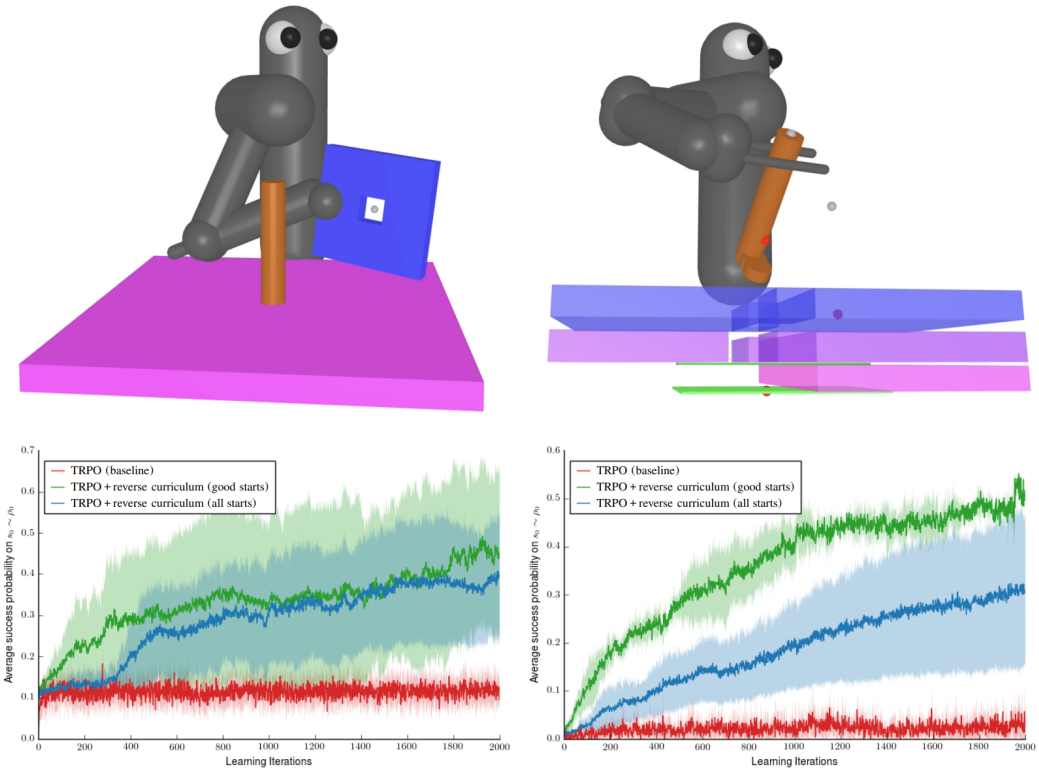
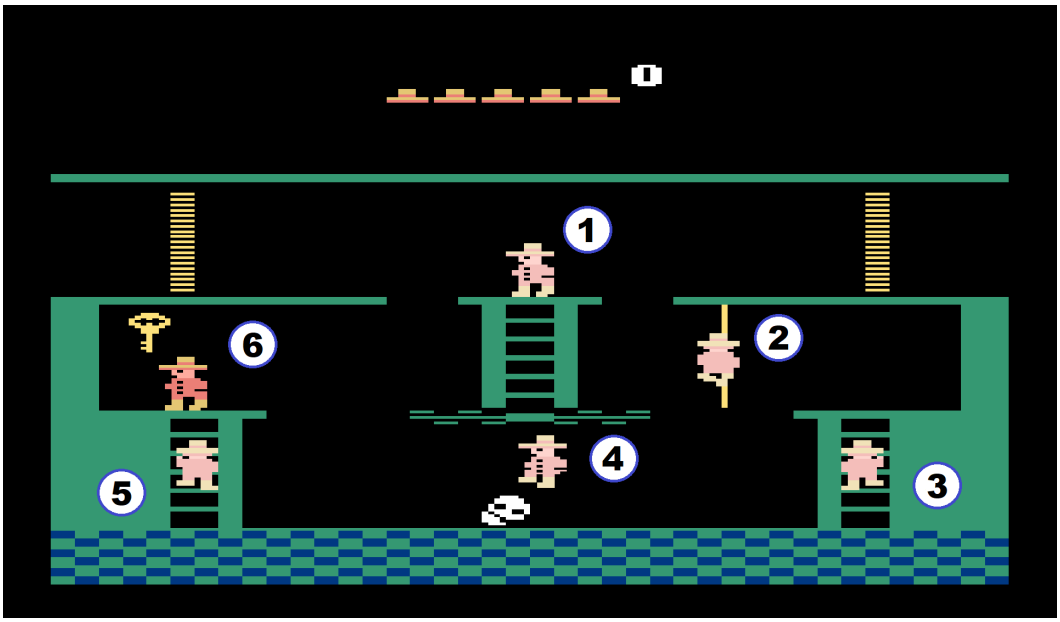


Figure 4.5. Results published by Florensa et al. [29], comparing their method against TRPO [87]. The tasks are: ring on peg (left) and key insertion (right). Their objectives are self-explanatory from their names; the former task requires a 7 DOF robot to place a square disk on a round peg (task is complete when the disk is within 3 cm of the bottom of the 15 cm tall peg). While the latter asks the same robot to execute a complex maneuver before properly inserting a key into a keyhole (i.e., it must first insert the key at a specific orientation, then rotate it 90 degrees clockwise, push forward, then rotate 90 degrees counterclockwise). In both cases, agents receive joint angles as inputs (the disk and the key are welded to the hand of the robot; hence, extra information about their poses is not necessary), and are positively rewarded only when finishing a task. Additionally, the original initial state distribution of both environments is uniform over their entire state spaces (this is what TRPO faces). The graphs at the bottom display the evolution of the success ratio as training progresses. They unveil that while an algorithm like TRPO, which lacks any mechanism for dealing with sparse rewards, is hopeless in these robotics tasks. But when the same technique is made to follow a reverse curriculum that exploits resettability and reversibility, its competence then skyrockets.

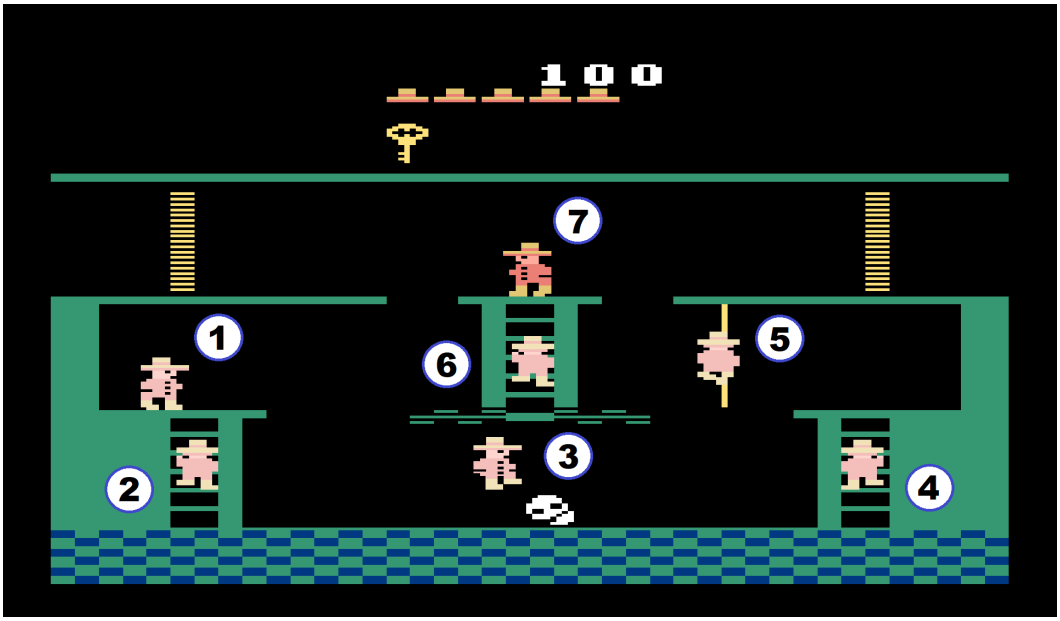
another seen afterwards if the avatar occupies the same spatial location in both. They are just too alike for this learner to tell them apart, and as a result, it assigns an equal bonus to both of them; which is true for every location along the path between the agent's initial position and the key. A direct consequence of this bonus conflation is that no round-trip policy can ever procure a higher return that the most optimal outbound policy. And this is ultimately the most likely cause of this learner never leaving the initial room; it has little to none motivation to retrace its steps. Conversely, see that when the avatar is given five lives instead of one as in the original setup, a second strategy for escaping the initial room becomes viable, which involves killing itself right after getting the key. An action that instantly places the avatar back to its initial position while still holding the key. Thus, as backtracking is no longer mandatory here, the same DQN-CTS learner is now quick to learn to commit tactical suicide and eventually ends up discovering around 15 rooms during a single run (see Figure 4.8). According to Dann et al. [18], the inability of the previous algorithm to differentiate between pre- and post-key observations even in the fully observable first room of Montezuma's revenge is due to the relatively small size of the key. Therefore, we should expect the problem of retracing steps to be more prominent in partially observable environments, which would not be conditioned by size in the same way (see Figure 4.7). On top of that, methods driven by across-training novelty generally map observations directly to bonuses (i.e., no internal states, no memory), lacking any mechanism that would otherwise allow them to temporally discriminate two observations within an episode.

4.3.4 Partially Ordered Subtasks

This feature is visible in domains that request the learning agent to carry out multiple distinct subtasks during a single episode, with many of them starting from the same state. An archetypal environment is one where the agent is initially located in a central spot from which a finite set of culs-de-sac expand radially outwards (see Figure 4.9). Each cul-de-sac imposes a different subtask as it urges the mastering of a specific ability (e.g., swimming, climbing) and the full task involves visiting every one of them all the way to the end. An important detail is that the environment provides no indication (inside an observation) about the completion of any subtask. Only when it completes the full task, the agent receives a reward. We say that the previous set of subtasks is partially ordered because the agent is free to attempt them in any order at any point in time. Nevertheless, the solution to the problem may or may not depend on the order in which these subtasks are performed. Stanton et al. [98] reason about various mechanisms by which domains comparable to the former example can pose a complication to across-training bonus-based algorithms [9, 74, 12]. First of all, for most such methods, if not all of them, their bonus function remains essentially unchanged in the short term (during a single episode). In other words, if the learning agent stands still, it will receive the same bonus at each time step. The consequence of this is that in the presence of partially ordered subtasks that do not signal subtask completion the optimal policy is always one that visits the most rewarding cul-de-sac and stays there until the episode terminates. Under such circumstances, it is unclear whether or not these algorithms are at all capable of learning a policy that concatenates diverse abilities in a controlled manner

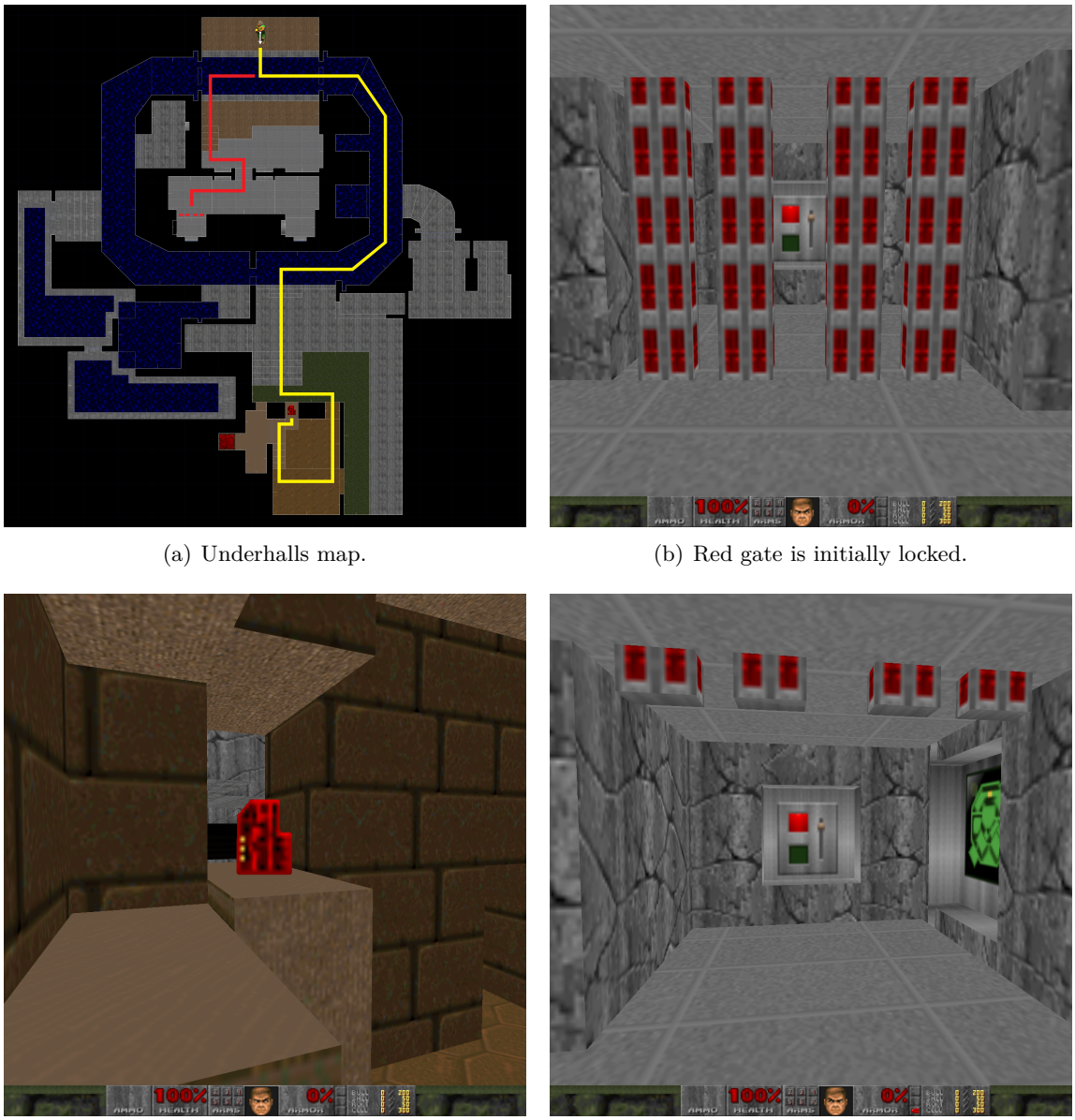


(a) Panama Joe's journey to the key on the left.



(b) Panama Joe's journey back to his starting position.

Figure 4.6. First room of Montezuma's revenge. To leave it, and if life loss must me avoided, Panama Joe has no choice but to retrace his steps after collecting the key that opens any one of the two yellow doors at the top.



(a) Underhalls map.

(b) Red gate is initially locked.

(c) Red card key is found.

(d) Red gate is opened with the red key.

Figure 4.7. The second level of the classic first-person (i.e., partially observable) game Doom 2, named Underhalls, requires the doomguy to unlock a red gate. As seen in the level's map, this gate is close to the doomguy's initial position (red path), but to open it he is forced to first travel forth and back through several other sectors of the world (yellow path) to retrieved a red card key.

within an episode. A second difficulty is catastrophic forgetting [32], which is when a neural model loses an acquired ability in response of not performing it recently. Across-training bonuses are usually designed to decrease over time as the learner keeps reaching the same states or repeating the same transitions (this happens in the three methods previously given as an example). In this context, environments with

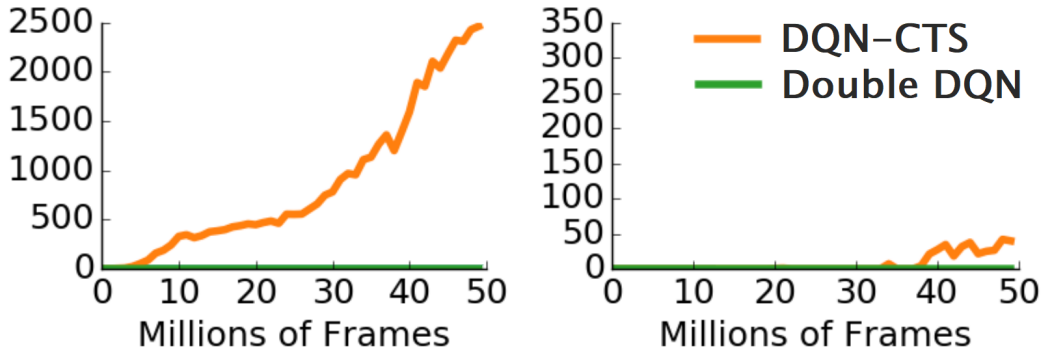


Figure 4.8. Scores achieved by a couple of algorithms in Montezuma's revenge when the avatar is given 6 lives (left) and one life (right) per episode (reported by Roderick et al. [83]). We see that in a difficult sparse-reward environment such as this one a technique like Double DQN [41], which has not been specifically designed to handle an overwhelming lack of reinforcement, proves to be utterly incompetent. In turn, a DQN-CTS agent works fantastically when backtracking is not compulsory (in the standard setup with 6 lives), but it barely gets any points when it must retrace its steps (in the single-life setup).

partially ordered subtasks are especially interesting since they open the possibility for such approaches (and in fact any approach) to focus on mastering a single subtask at a time. Assuming that happens, these algorithms may for instance choose to dedicate themselves solely to learning how to swim at the beginning of training, becoming good at it after some time. At that point, the bonuses accredited to swimming would have decreased so much that it would be more rewarding to stop doing it entirely and start focusing on another activity, like climbing. And here is where catastrophic forgetting becomes a problem, it would obviously take these learners another significant amount of time to pick up climbing, and by the time they do so they would have forgotten all about swimming. Ultimately, catastrophic forgetting can be expected to push these algorithms to become specialists, which perform one subtask with competence and are clueless about every other one. In addition, catastrophic forgetting would also affect the exploration models of such methods, make them lose memory of having visited every other cul-de-sac but the one currently being paid attention to. One last challenge put by environments displaying partially ordered subtasks arises when they additionally demand performing them in a specific order to get a reward. For this analysis, let's also consider that the prior issues of learning a composite policy and catastrophic forgetting have been resolved. In such scenario, if we again assume that methods guided by across-training bonuses can fixate on one subtask at a time, then it would be possible though undesirable for them to focus on the wrong subtask first. For example, while a problem requires to first go swimming and then go climbing, these learners may unfortunately decide to master climbing first. After becoming proficient climbers, they will start getting interested in swimming, which will concentrate higher bonuses by now. Given our initial premises, these algorithms will then be driven to concatenate climbing and swimming, in that order, within the same policy and episode. Thus, they will learn to swim while still going climbing at the start of every episode. And that means that

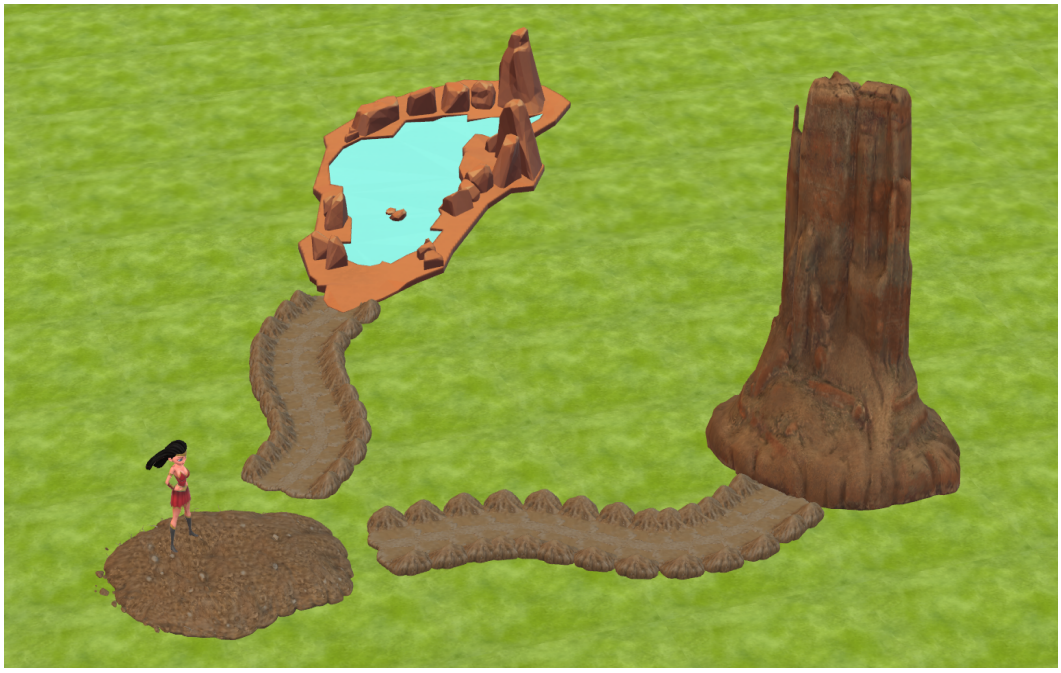


Figure 4.9. Idealized scenario that consents an agent to freely explore different sub-tasks, such as swimming or climbing.

from this point onward the visitations and bonuses associated with climbing will never become less and higher, respectively, than those associated with swimming. In the end, these methods will not have enough incentive to go climbing a second time after swimming and the full task will not be solved.

4.3.5 Distractors

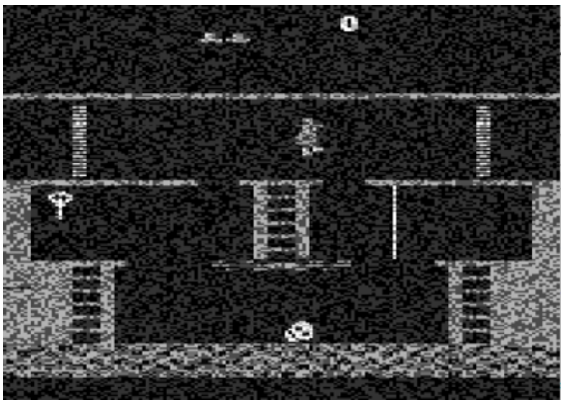
A distractor is a dynamic element that can be observed by the learning agent but is irrelevant to the resolution of the task in hand. Distractors come in different flavors: stochastic or deterministic (usually with extremely complex or even chaotic dynamics), controllable or uncontrollable. Pathak et al. [74] offer the scenario of an agent observing the movement of tree leaves in a breeze as a very compelling mental image of a distractor. The motion of millions of leaves is undoubtedly a complex dynamical system that has an enormous number of feasible configurations and whose dynamics is impossible to predict from purely visual information. On top of that, the agent cannot manipulate the wind or the leaves directly; hence, such observed dynamicity transpires entirely beyond its control. Figure 4.10 portrays other examples of distractors. Generally speaking, distractors introduce two important complications into a reinforcement learning problem: the addition of plenty of purposeless states and transitions as well as the unpredictability of their dynamics. Given the generality of these two concerns, distractors have the potential to affect virtually any method known to tackle sparse rewards. Notwithstanding, all experimental evidence found in the literature regarding the detrimental presence of distractors is in reference to bonus-based approaches. It is clear why these methods experience such hardship when facing distractors. To circumvent reward

sparsity, they generate dense bonuses grounded on the idea of rewarding agents for finding or learning something new in the environment (e.g., novelty, curiosity). Therefore, since distractors can be designed to produce endless new observations and to embody unlearnable dynamics, they can easily attract the complete attention of such algorithms and make them ignore the actual task in full. For example, Kim et al. [49] as well as Song et al. [96] add pixel-level white noise over each image observation received by an agent playing Montezuma's revenge (see Figure 4.10(a)). In addition, the dynamics of this noise is kept independent from the agent's actions (i.e., it cannot turn it off or relocate it). Results from both these works show that the performance of RND [12], an across-training novelty-driven approach, falls sharply to nearly zero when such uncontrollable stochastic distractor is in place (see Figure 4.11(a)). And these two studies agree that such degradation is likely a consequence of the continuous stream of new noise configurations, which artificially maintains novelty high in every state, giving the leaner no motivation to travel far in this world. Another distractor with similar characteristics is proposed by Pathak et al. [74]. In this case, an agent is asked to solve a sparse-reward navigation task in a visually rich 3D environment. And the distractor is a large region of white noise stitched next to each image observation sent to the agent, which again has no control over the distractor. The study experiments with an exploration policy guided by an across-training curiosity bonus, computed directly as the prediction error of a learned one-step forward dynamics model (i.e., a model that given the latest observation and action predicts the next observation). And this assessment concludes that such policy fails to attain the maximum return in the problem detailed above. The authors argue that the impossibility of predicting white noise with zero error keeps the curiosity bonus slightly up everywhere and this discourages the exploration policy from leaving its current location in the environment. Burda et al. [11] along with Savinov et al. [84] investigate the effects caused by controllable stochastic distractors. Both works adopt a visual navigation domain, which is augmented with a noisy TV that functions as a distractor. This TV is implemented as a region of the environment that always displays some picture taken from a large dataset. In the first work, this TV is present only in one location of the environment (a wall whose view can be blocked by others); while in the second, it is placed on top of every image observation, covering the same area in each interaction (as shown in Figure 4.10(b)). Importantly, in either case, the learning agent is endowed with one dedicated action that allows it to change the station of this noisy TV at will. When the agent chooses to do so, the domain replaces the current TV picture with another taken uniformly at random from the corresponding dataset. The two studies evaluate the response of ICM [74] to this noisy TV. To incite exploration, ICM relies on an across-training curiosity bonus that, exactly as in a method mentioned before, is equal to the prediction error of a forward dynamics model. But contrastingly, the observation embeddings going into that model are shared with an inverse dynamics model trained in parallel (i.e., a model that given the current and next observations infers the action taken). Because of that, those embeddings encode information of only elements the agent can control, which makes ICM immune to uncontrollable distractors (as proven by its authors). In the end, the findings of these works are consistent with each other, and demonstrate that ICM is not suitable against a noisy TV (see Figure 4.11(b)). This is explained by noticing that information about the

noisy TV necessarily leaks into the shared embeddings since it improves the accuracy of the inverse dynamics model, and that negates all the benefits of incorporating this extra model. In other words, ICM will become aware of this TV and it will be strongly attracted to its constantly surprising transitions. Catacora et al. [13] note that previous works have heavily relied on the difference in dynamics between distractors and their encompassing tasks to design algorithms tailored to ignore dynamic elements that are uncontrollable or have highly stochastic dynamics. To blur the line between distractor and actual task in terms of dynamics, this new study proposes two benchmark environments containing controllable deterministic distractors. In both cases, once again, the main task involves visual navigation and the distractor is a TV that covers the whole field of view of the interacting agent, which is given one action to turn the TV off and on. One TV is rendered as an 8 by 8 grid where each cell can take any one of 16 colors (see Figure 4.10(c)). Seeing that the total number of TV stations is immense, the agent is additionally provided with 4 special actions that allow it to navigate to any station within an episode. Furthermore, the dynamics of this distractor is chaotic in the sense that starting from the same TV station two action sequences disagreeing only in the first one will generate two completely different sequences of stations. When RND and ICM are tested on this environment, they both neglect the navigation task and fixate purely on the TV. These two approaches learn rather simple policies that move straight forward in some general direction of the TV space while almost never observing the same station or transition twice during training. And that is thanks to the stochasticity of the policies themselves and the chaotic regime of this distractor. The second TV is rendered as a 4 by 4 grid where each cell can be one of two colors. Here, the agent is granted just one extra action to interact with the TV; thus, it can ever experience only one sequence of stations. The online version of EC [84] is evaluated in this second scenario. EC incentivizes exploration by means of an intra-life curiosity bonus. In a nutshell, EC learns a reachability model that predicts if two states are less or more than a predefined number of actions apart from each other. With the help of that model, in each episode it keeps a memory of states that when first encountered are far from every state already in memory and assigns a large bonus to them. Such distance constraint enables EC to successfully handle a noisy TV (highly stochastic dynamics). Unfortunately, when EC is confronted with the second deterministic TV, it becomes utterly mesmerized by it. The reason is clear, EC quickly learns the trivial policy of always changing the station, which collects several curiosity bonuses, but it is merely a local maximum.

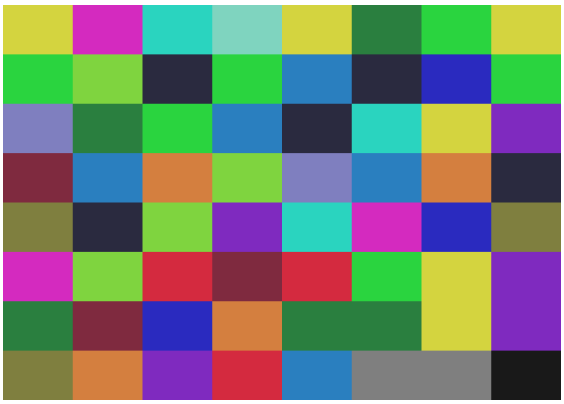
4.3.6 Stochastic Dynamics

As seen in the previous subsection, the existence of elements with stochastic dynamics can be a serious inconvenience for some exploration strategies. Sadly, such elements are not limited to act as distractors, and indeed, they can be an inextricable component of the task we wish to solve. A good example is the introduction of sticky actions in games of the Arcade Learning Environment [63]. Sticky actions is implemented simply by defining that the action ultimately executed by an avatar is chosen probabilistically between the one commanded by the learning agent and the action performed in the preceding interaction step (see Figure 4.12). This

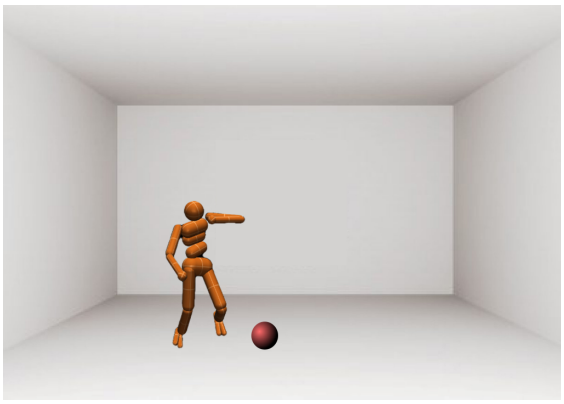


(a) White noise applied to Montezuma's revenge.

(b) Noisy TV displaying animal pictures.



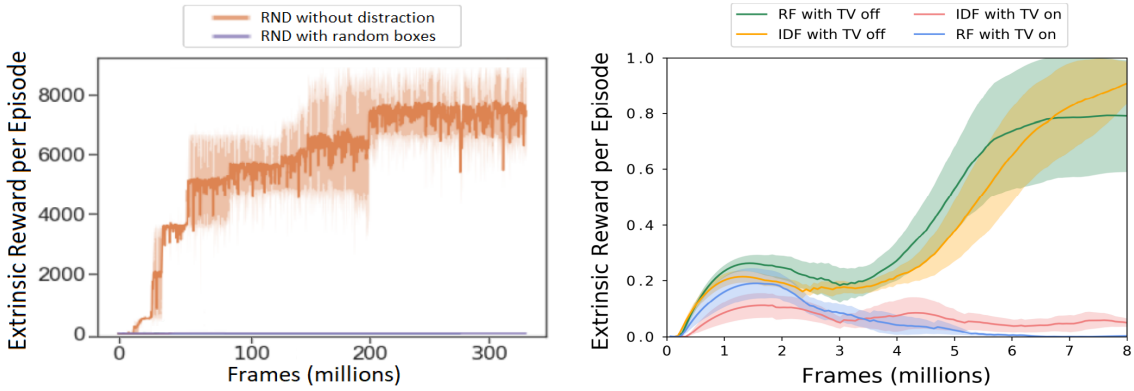
(c) One station of a controllable deterministic TV.



(d) An ordinary ball can be a powerful distractor.

Figure 4.10. Miscellaneous distractors.

means that, from the learner's perspective (which is only ever aware of the action it instructed), facing sticky actions is equivalent to facing an ubiquitous stochastic distractor, i.e., one that partakes of every transition. And a direct consequence of this is that when sticky actions are employed it is impossible to learn a perfectly accurate forward dynamics model of any part of an environment. Burda et al. [11] express their concern regarding the limitations under stochastic dynamics of across-training curiosity-driven methods, which rely on the prediction error of a forward dynamics model to compute the bonuses that help them overcome reward sparsity. These include algorithms built on a rather straightforward usage of the prediction error [97], but also those applying more elaborated schemes such as ICM [74]. As stated earlier, the unpredictability of the world is detrimental to this family of methods, because it can caused curiosity bonuses everywhere to remain high, which eventually may hinder the development of far-reaching exploration policies. In practice, these learners will still visit locations distant from the environment's initial state, especially when the probability of an action to get stuck is low. Yet its overall performance in terms of coverage and sample efficiency will degrade considerably. That stochastic dynamics and sticky actions specifically have a negative impact on a technique along the lines of ICM is emphatically validated by Figure 4.14.

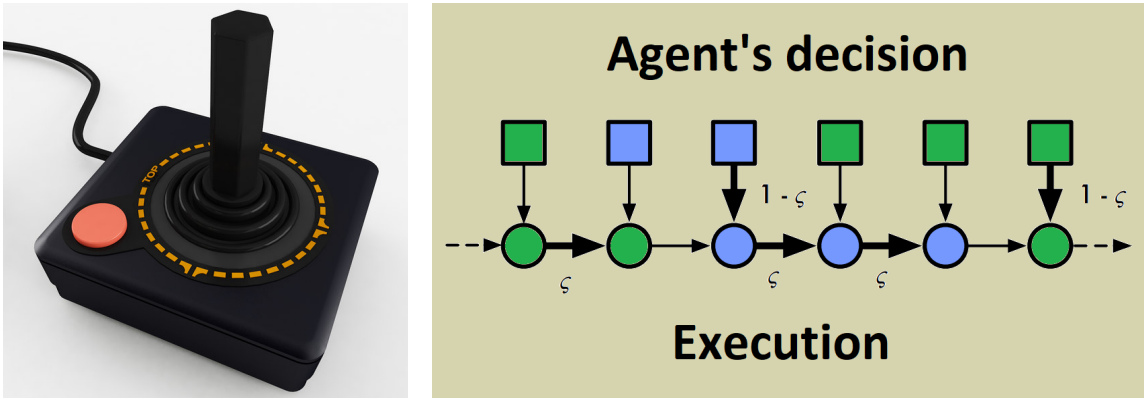


(a) Montezuma's revenge + random boxes.

(b) 3D Maze + noisy TV.

Figure 4.11. Two experiments demonstrating that algorithms developed to work under sparse rewards are sometimes troubled by distractors. The one on the left, run by Kim et al. [49], evaluates RND in Montezuma's revenge without and with an additional distraction. The latter is framed as 7x7 boxes filled with pixel-wise noise that can randomly pop up anywhere within each image observation. The presented graph convincingly exposes that, even though this algorithm does fairly well in the standard setup of the game, it just cannot handle an uncontrollable stochastic distractor. Meanwhile, the experiment seen on the right, described by Burda et al. [11], examined a pair of methods that harness curiosity as a way to improve exploration. One is in fact ICM, which, as we know, extract features from observations with the help of a trained inverse dynamics model (thus, these features are called IDF). The other simply replaces the observation embeddings of ICM with random features (RF), which are generated by a neural network whose weights are initialized at random and then are kept fixed the entire training session (they are never updated). Both these techniques were tested in the same static 3D maze twice (containing a single task reward of +1 given at the goal), once without and another time with a noisy TV. The results of these 4 runs leave no doubt about the detrimental effects of this TV on these methods. Without it, they comfortably learn a policy capable of reaching the goal most of the time; yet, when such distractor is present, they appear completely lost.

There we see the outcomes of two sets of experiments, executed as part of this thesis, in which said algorithm was asked to solve the maze navigation problem depicted in Figure 4.13. In half of the trials, the avatar was bounded to always enact the actions commanded by the learning policy (non-sticky scenario) and, in the other half, it sometimes ignored such orders and repeated its previous move instead (sticky scenario). Critically, in both experimental setups the environment as well as the learner were configured in exactly the same way. Every single hyperparameter was identical except obviously for the probability of sticky actions taking place, which was set to 0.0 and to 0.25 in the former and latter scenarios, respectively. These results evidence that the advanced exploration capabilities offered by ICM are categorically undermined by sticky actions. In their absence, this technique worked as desired; it discovered all rooms and sectors of the maze while mastering a policy that arrives at the goal every time. Yet, in the sticky setup, not only it never found the jewel, but it also barely made it out of the initial room; meaning that around 80% of the world remained uncharted even after 40 million frames of experience.



(a) Atari joystick.

(b) Visual representation of sticky actions.

Figure 4.12. The implementation of sticky actions emulates a well-known issue occurring in physical controllers where buttons have a tendency to get stuck. As the diagram shows, given two actions (green and blue), at each time step either the one selected by the agent (squares) get executed or the one from the previous step (circles) is repeated, with probabilities $1 - \zeta$ and ζ respectively.

4.3.7 Action-Prediction Degeneracy

The problem of predicting which one action caused a given transition is said to be degenerate when there is no single answer. Such situation emerges in environments whose dynamics permit more than one primitive action to produce the exact same transition. The classical example is that of an agent pressing a solid block straight into the ground or against a sturdy wall; no matter what the strength of the applied force is, the block will not move (see Figure 4.15). This kind of degeneracy becomes a worrying issue for any algorithm that internalizes an inverse dynamics model anywhere within the machinery that allows it to thoroughly explore sparse-reward environments. That is, a model that given a sequence of states or observations gathered from a past trajectory tries to postdict the actions performed by the interacting agent at each transition. Models of this sort have been employed in previous works for different purposes. For example, Haber et al. [39] use one such model to generate a dense curiosity bonus that guides exploration. Specifically, each transition is assigned a bonus reward equal to the prediction error of an inverse dynamics model that is learned alongside an interacting agent. This motivates such agent to visit those regions of an environment where its control is less understood. Moreover, since this model is continuously updated from trajectories executed by the interacting agent, the bonus of a region will tend to decrease as visitations to that region increase. In an ideal scenario, this exploration mechanism would explore an entire environment, starting with regions closer to the environment's initial state and moving to further away regions as the bonuses of the former ones drop to zero. Unfortunately, that idyllic vision is shattered when degeneracy renders it impossible to predict with perfect accuracy which action provoked an observed transition. That keeps bonuses relatively high all over the world (even near the initial state), which ultimately disincentivizes the agent from exploring remote locations of the environment. Pathak et al. [74] also compute a curiosity bonus that depends

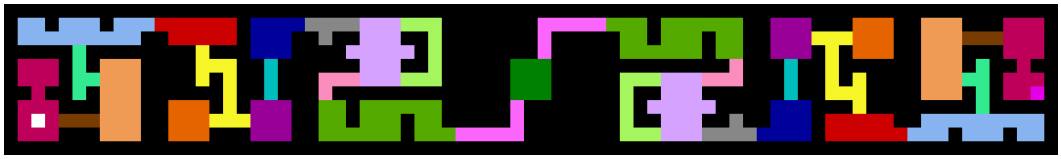


Figure 4.13. Environment used for testing the effects of sticky actions on ICM. At its core, this domain is a maze structured as a 11x77 gridworld. It is further divided into seven 11x11 rooms and 35 sectors (groups of adjacent cells of the same color). The task to be completed here is the following. Starting randomly from any cell within the carmine sector of the leftmost room, the avatar (single white cell) must navigate every room and at least 31 sectors of the maze to collect the jewel (single magenta cell) located in the rightmost room. Upon taking this gem, the learning agent is handed a reward of +1 and the episode is terminated. No other extrinsic reward exist in this domain. Plus, if the jewel remains out of reach, the ongoing episode automatically ends after 1000 time steps. To interact with this world, the agent is allowed to see only the room it is currently in (i.e., it has partial observability). In particular, every observation it receives is a 44x44 image, which means that each cell of the maze is represented as a 4x4 patch of solid color. In terms of actions, the avatar is entitled to travel exactly one cell up, down, right or left per time step (4 discrete moves in total). There is no noop action and, furthermore, to collect the jewel, the avatar just needs to step over it. Importantly, if the agent tries to move into an impassable wall (black cells), nothing happens, it simply stays in the same cell.

on a one-step inverse dynamics model. However, this time the prediction error of such model has no direct influence in the value of the bonus. Instead, this bonus is proportional to the prediction error of another forward dynamics model, which is trained concurrently with everything else. And the role of the inverse model is purely to ground a representation shared by both models. Again, a degenerate action-prediction problem can have some impact here, since this algorithm has no explicit mechanism that would negate the occurrence of a fluctuating biased policy. That is, a policy that favors some actions over others at degenerate transitions and whose preference flip-flops from one update to the next (as doing so changes nothing). Ultimately, the variability in the training data induced by such a biased behavior would cause all models to converge slower. In other words, the inverse dynamics network, its representation and the policy itself would take longer to settle down than otherwise would in a non-degenerate domain of similar complexity (e.g., with the same number of actions or examples).

4.3.8 Deadly States

This feature's definition is quite self-explanatory from its name (a couple of examples are shown in Figure 4.16). Some environments simply possess states where an episode immediately terminates if the interacting agent visits them, and such termination can happen much earlier than the episode's timeout. Furthermore, in sparse-reward environments, the agent usually gets no reward when falling into these deadly states. The presence of these states can strongly disrupt the exploration behavior of various approaches created to handle reward sparsity. Indeed, it should not be surprising that a learner that easily covers the entire state space of a domain without deadly states fails to do the same or works much more inefficiently when such states are added.

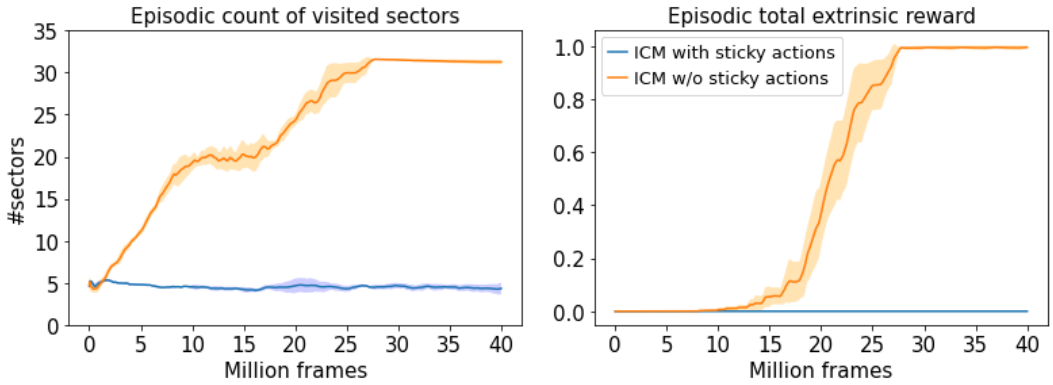


Figure 4.14. Evaluation of ICM in the sparse maze navigation task shown in Figure 4.13, in both the presence and absence of sticky actions. For these experiments, the implementation of ICM replicated the setup employed by Pathak et al. [74], but with the upcoming modifications. During preprocessing, grayscale conversion and frame stacking of 4 were still applied (to produce 44x44x4 observations); yet, no action repeat was enforced. The last layer within the embedding networks of both, the policy and the curiosity module, was assigned a stride of 1 and no padding, which resulted in representation vectors of dimensionality 512. Accordingly, the hidden layers of the forward and inverse dynamics models as well as the output layer of the former were also enlarged to 512 units. Within the policy, the previous embeddings were sent, rather than to an LSTM, to a stack of 4 dense layers, each with 512 neurons and ReLU activation. PPO was used as base learner while being configured as follows: 16 workers, 4 epochs per update, 4 mini batches per epoch, rollout length of 80 steps per worker, γ of 0.99, λ of 1.0, clipping ϵ of 0.05, value function coefficient of 0.5 and entropy coefficient of 0.01. Optimization was carried out by Adam with learning rate of 5×10^{-5} and maximum gradient norm of 40. Lastly, all hyperparameters of the curiosity module remained unchanged except for β, which was lowered to 0.005. The graphs shown here reveal what happens when the previous instantiation of ICM is tested in the abovesaid maze environment under two regimes: without sticky actions and with them (with probability ζ=0.25). Both scenarios were run 3 times, each with a different seed. Correspondingly, these curves report means and variances across those 3 runs of simple moving averages over 5 million time steps computed at the level of each individual training session. These results demonstrate a stark contrast in the performances of ICM due to sticky actions. When the latter were not integrated into the task, ICM managed to learn a strategy that gathers the jewel in nearly every episode. Conversely, in the sticky scenario, this same algorithm not once in 40k episodes landed on said jewel. Even worse, on average it frequented merely 5 sectors of the maze (i.e. just the entire initial room), leaving the remaining 6 rooms and 30 sectors mostly unexplored (barring a few short-lived shallow incursions into the second room).

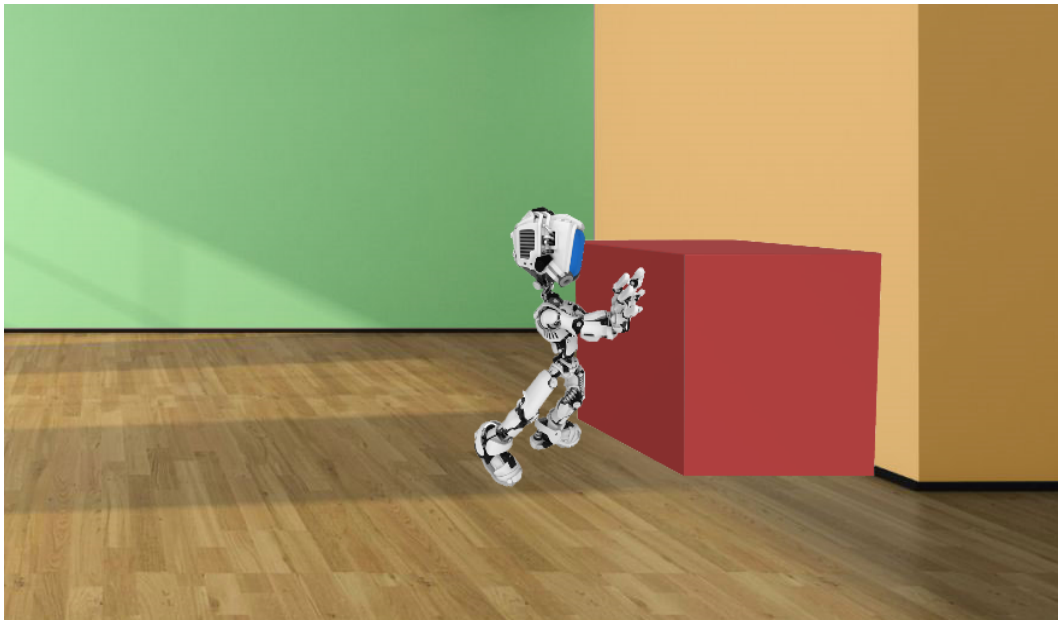


Figure 4.15. Robot hopelessly pushing a solid block into a thick concrete wall. Even as it varies the magnitude of the exerted force, no effect is perceptible.

For instance, Dann et al. [18] present the notion of risky exploration. This situation occurs when an environment expects the learning agent to acquire a complex skill (i.e., a sequence of a few actions highly unlikely to be performed by chance), but small deviations from a perfect execution lead to life loss. The authors argue that such setting appears, for instance, as an agent tries to jump over the first moving skull in Montezuma's revenge, which additionally requires the agent to move towards the deadly skull before doing the jump. According to them, risky exploration poses a serious challenge to approaches relying on across-training reward bonuses that decrease in response to increasing visitations (e.g., [9, 74, 12]). For these agents, seeing that they internally receive a positive reinforcement for every time step they survive, dying also means that they will miss out on collecting a sizable cumulative bonus. Therefore, they have a big incentive for staying alive, even if that means putting the surveying of the environment on the back burner. Going into more detail about the mechanics of it all; as these algorithms gradually explore their environment, states or transitions lying just before a recently discovered life-threatening obstacle will necessarily be associated with higher bonuses. Consequently, such learners will be motivated to re-visit that deadly obstacle. Then, after a few failures they will realize that the obstacle is not easily surmountable and that their best local policy is to reach its edge and then sit tight, taking no risky actions while soaking up the bonus of that zone. But as visitations to the edge increase, the bonus there will get depleted, until eventually it will be more rewarding to stop a little before the edge and soak up the bonus of preceding states or transitions. This process will repeat over and over again, pushing these learners further and further away from the deadly obstacle. Still, at some point, these agents will become interested again in the obstacle and the cycle will restart. All in all, the phenomenon of risky exploration induces, at best, a highly inefficient exploration, and at worst, a decisive impasse in

(a) Lava rivers in Doom.

(b) Abysses and Bullet Bills in Super Mario Bros.

Figure 4.16. Examples of states appearing in environments commonly employed by the reinforcement learning community that produce the death of an interacting agent.

which deadly states are never overcome because of a shortage of experiences.

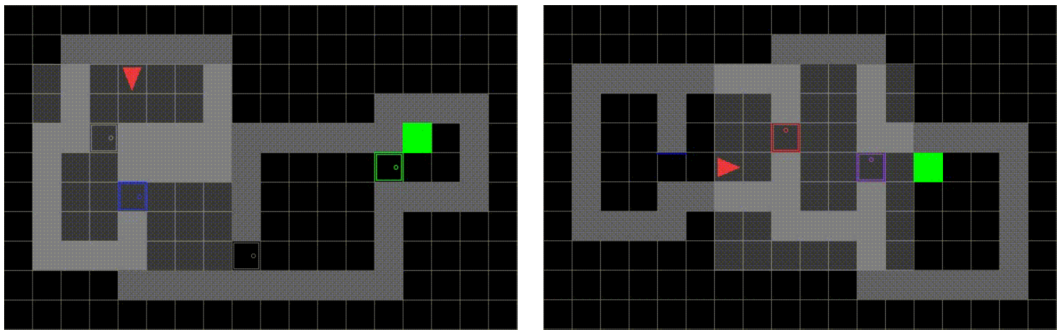
4.3.9 Procedural Generation

Some environments dictate that every episode is to present a drastically different scenario to the interacting agent in terms of: map configuration (geometries, textures, colors, etc.), existing elements (keys, doors, deadly states, distractors, etc.), dynamics, among other attributes. But that, at the same time, all episodes are to convey the same task at an abstract level (e.g., maze navigation). In practice, this is achieved by means of a dedicated procedural generation code that randomizes over a list of attributes and their parameters to create a unique episode every time. Because of this randomization no two episodes will be alike in these environments (see Figure 4.17). And in addition, it will be highly unlikely to observe the same state or transition twice throughout training. If on top of all that such environments also have sparse rewards, then we get a very ambitious and exciting problem: learning exploration strategies with generalization capabilities. Presently, since these domains have been little studied so far, it is unclear whether or not existing algorithms of any flavor are well-suited for dealing with them. For instance, learners seeking new experiences across training episodes, such as ICM [74] and RND [12], might not be motivated to do deep exploration as every initial observation or transition is already brand new by default. Several algorithms that develop an automatic curriculum are also at odds with procedurally generated environments. Particularly, those that tell the learning agent which goal to reach at the beginning of each episode and that select such goal from a buffer of previously seen states (e.g., [109, 24]). Clearly, if all episodes are different from each other, the technique of using past states directly as goals will mostly produce unattainable ones.

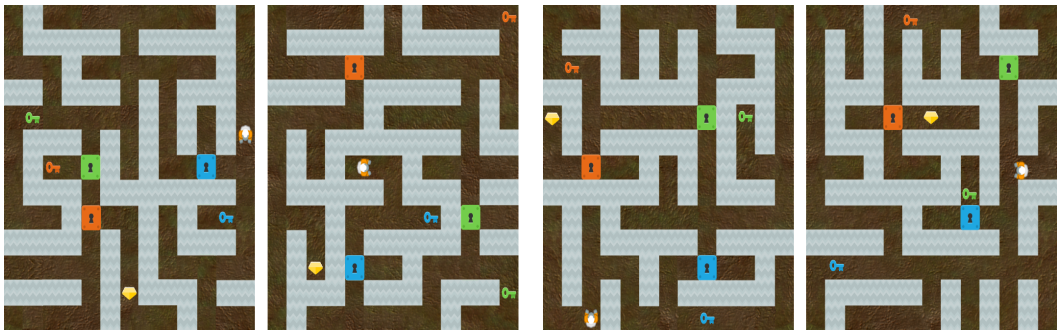
Despite these adversities, a handful of works have already experimented with such sparse-reward procedurally generated environments, and they have even proposed some promising solution methods. For example, EC [84] sees a continuous increase of episodic rewards during training in a partially observable 3D visual navigation task, in which the map layout and the textures of walls and floors change between episodes. Notably, such increase rises significantly higher and is more consistent across runs than what it is observed for other algorithms, namely PPO [88] and ICM [74]. As described in Section 4.3.5, EC uses a learned action distance classifier (that takes two states or observations as input) together with a memory buffer to give itself bonuses whenever it reaches states that are far from every other state visited earlier during the current episode. Arguably, as long as EC is able to generalize such distance classifier, it should do well in any given procedurally generated environment. Luckily, in many domains this is actually feasible and not necessarily hard. Like in the one tested by the authors, where observations belonging to rooms with alike and different textures could be taken to be close to and far from each other, respectively. A similar solution concept was employed by Badia et al. [77] to develop NGU, which thoroughly explores fully observable 2D mazes of varying contours from visual inputs. NGU combines two exploration bonuses in a multiplicative way. One rewards novelty across episodes, and it is implemented simply by invoking RND without any extra modifications. Meanwhile, the second bonus rewards episodic novelty. Its computation is grounded on a learned controllable representation (similar to ICM). And, to put it in very simple terms, the bonus assigned to each observation is inversely related to the sum of the similarities measured between its representation and those of all previously seen observations within the current episode. More succinctly, large bonuses correspond to observations that are less similar to every other one occurring earlier in an episode. A few extra details: similarities are computed according to a predetermined kernel function (non-learnable); their sum considers only the k-th nearest neighbors; and past observations are stored in an episodic memory buffer. RIDE [79] shows a good performance in a variety of partially observable 2D visual navigation domains, whose topologies (rooms dimensions, locations of passages between rooms) and dynamic elements (positions of keys, doors, boxes, etc.) are readjusted across episodes. RIDE learns the same forward and inverse dynamics models as ICM, alongside a shared representation. However, its bonus, named impact-driven reward, is instead defined as the Euclidean distance between consecutive state representations, divided by the episodic visitation count associated with the next state. In large or continuous observation spaces, this count can be estimated thanks to a density model as done by Bellemare et al. [9]. Again, just like the two previous methods, RIDE is built around an episodic bonus that will induce a steady coverage-promoting intrinsic reward landscape once its latent representation stabilizes. A landscape that, in addition, will be perfectly tailored to each procedurally generated episode of an environment. Another promising algorithm and one that deviates substantially from the former ones is AGAC [26], which has been successfully tested in the same benchmarks as RIDE. AGAC trains a policy and a value network, as various other methods do, but critically it also trains an adversary network. The objective ascribed to this adversary is to minimize the Kullback-Leibler divergence between its own and the policy's action distribution. Meanwhile, the policy is set to maximize task reward, as usual, and it also tries to maximize its discrepancy with respect to the adversary (i.e.,

difference of action log-probabilities). According to the authors, this formulation is helpful in sparse-reward settings because it fosters the diversity of trajectories as the adversary constantly pushes the policy to behave differently from past executions.

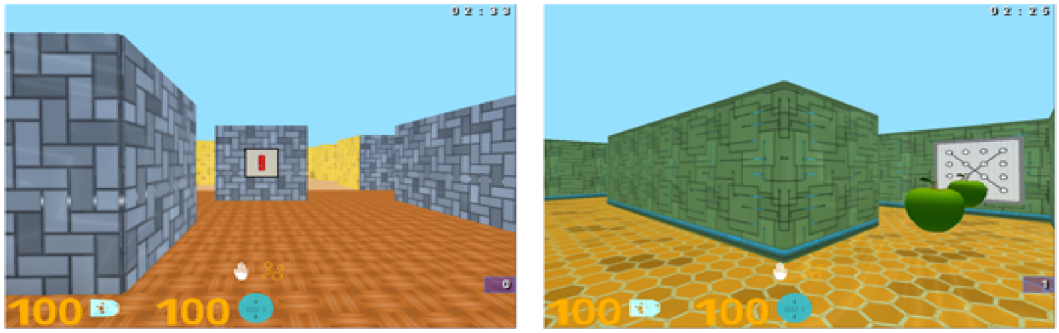
Currently, there is a meaningful amount of stimulating benchmark environments that integrate sparse rewards and procedural generation. At this point, it is important to make one clarification. To categorize a procedurally generated environment as having sparse rewards, it is not enough that it has few rewarding states in every episode. It is also imperative that in any episode generated by such domain the chances of a simple policy (e.g., one that selects actions at random) finding a reward are extremely low. Otherwise, if the distribution of episodes ranges smoothly from those where the learning agent obtains a reward easily to those where this is a herculean feat, then this may create a learning curriculum that can be exploited even by algorithms lacking advanced exploration mechanisms (as noted in [74, 84]). That being said, the following is a list of sparse-reward procedurally generated benchmarks used in previous studies. MiniGrid [14] provides various 2D gridworld navigation tasks, where the interacting agent receives partial egocentric views of the environment (readily convertible into RGB images). The tasks more frequently employed by researchers are: MultiRoomNXSY, KeyCorridorSXRX and ObstructedMaze. MultiRoomNXSY rewards a learning agent upon arriving at the other end of a chain of rectangular rooms connected to each other by means of single-cell unlocked doors. The main attributes that change from one episode to the next include the dimensions and the relative position of each room as well as the placement and color of each door (see Figure 4.17(a)). KeyCorridorSXRX and ObstructedMaze also require the learning agent to traverse multiple rooms until finding a singular rewarding goal location (always found in a distant room). Additionally, these scenarios demand slightly more complex interactions, such as: grabbing a key from a box to open a door or pushing away a ball that is obstructing a door. In both cases, the map geometry does not vary between episodes, unlike the positions of keys, doors, boxes, balls and goal. Procgen [16] is a benchmark suite containing 16 environments, all implemented as 2D worlds with videogame-like visuals. Given that Procgen has been conceived primarily for assessing generalization, only a couple of environments have sparse rewards. A particularly interesting one is Heist, in which the learning agent must travel through a network of locks and keys encased within a maze until finding a gem. Procedural generation controls the contour of the maze plus the position of all items, as shown in Figure 4.17(b). The NetHack learning environment [51] is a testing platform framed around the terminal-based dungeon-crawler game of the same name. In this game, a hero is tasked with retrieving the Amulet of Yendor. To do so, it must descend over 50 procedurally generated levels of a dungeon and escape from it afterwards. The game's dynamics is profoundly complex as it is partially observable and comprises 93 actions along with hundreds of unique elements to interact with. Many aspects of the game are readjusted across episodes, including the dungeon's topology and the exact location of places and items of interest. DeepMind Lab [6] offers several visually rich 3D navigation tasks to reinforcement learning researchers. One readily available domain presents the learning agent with a new 3D maze to solve in each episode. The agent perceives the world through a first-person view camera (which can be occluded by walls) and receives a reward only when it reaches a goal location. Mazes



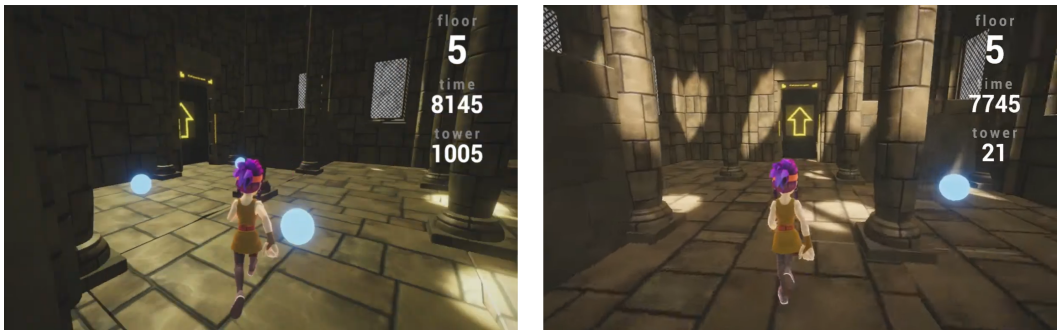
(a) MultiRoomNXSY (MiniGrid)



(b) Heist (Procgen)



(c) Initial observation produced by the random_maze domain (DeepMind Lab)



(d) Entrance to the 6th floor (Obstacle Tower)

Figure 4.17. Side-by-side comparison of procedurally generated episodes from various domains.

created in separate episodes differ in their geometry, the agent's initial position, the goal location and the textures applied to various sectors of the world (see Figure 4.17(c)). Likewise, Obstacle Tower [46] is a 3D environment with high visual fidelity, whose gameplay mimics multi-level adventure videogames. It consists of up to 100 floors that the learning agent must complete. Moreover, each floor is composed of multiple rooms and each room can contain some taxing subtask, such as: solving a puzzle, defeating enemies, evading obstacles or finding a key. As information, the agent is given images taken by a third-person camera that follows it everywhere plus an auxiliary vector encompassing abstract variables (e.g., keys in its possession, remaining time). Researchers are permitted to choose the reward structure of the environment; either sparse (+1 per completed floor) or dense (additional +0.1 for solving subtasks). Finally, as seen in Figure 4.17(d), each floor of Obstacle Tower contains procedurally generated elements, namely: lighting, textures, room layout and floor plan.

Chapter 5

Benchmarking Exploration-Intensive Distractors

Sparse-reward environments are notoriously difficult for deep reinforcement learning. Nonetheless, in response to such a challenge, researchers have recently developed numerous DRL algorithms able to handle reward sparsity to some extent. Not only that but some methods have even gone one step further and have tackled sparse-reward tasks involving different kinds of distractors (e.g., a broken TV, a self-moving phantom object and many more). This chapter puts forward two motivating new sparse-reward environments containing the so-far largely overlooked class of exploration-intensive distractors. Furthermore, a benchmarking is conducted that reveals that state-of-the-art algorithms are not yet all-around suitable for solving the proposed environments.

5.1 A Thin Line Between Task and Distractors

Despite the several breakthroughs achieved by state-of-the-art algorithms in sparse-reward domains, as of yet none can be said to be the definitive solution to every problem. For example, RND has been shown to fundamentally fail in the presence of pixel-level noise [49, 96]. Such noise is more generally referred to as a distractor, i.e., a dynamic element that can be observed by the learning agent and is irrelevant to the resolution of the task at hand. Distractors have also taken the form of: a broken TV [84], a remote-controlled noisy TV [11] or a self-moving phantom ball [55]. Crucially, the previous distractors share one commonality, their dynamics are easily distinguishable from those of most DRL benchmark environments. In particular, they possess at least one of the following three perverse properties. 1) They are uncontrollable; i.e., agents are conferred no actions that alter the state of these distractors. 2) They are highly unpredictable; i.e., their transition probabilities tend to adopt a uniform distribution, and hence, numerous and widely different states are equally likely to follow any given state. 3) They demand a low transitioning effort [84]; that is, moving between any two states requires a minimum of one time step. Unsurprisingly, previous works have leveraged these visible differences to formulate rather successful algorithms that first detect and then ignore elements whose dynamics display one of the former properties [74, 84].

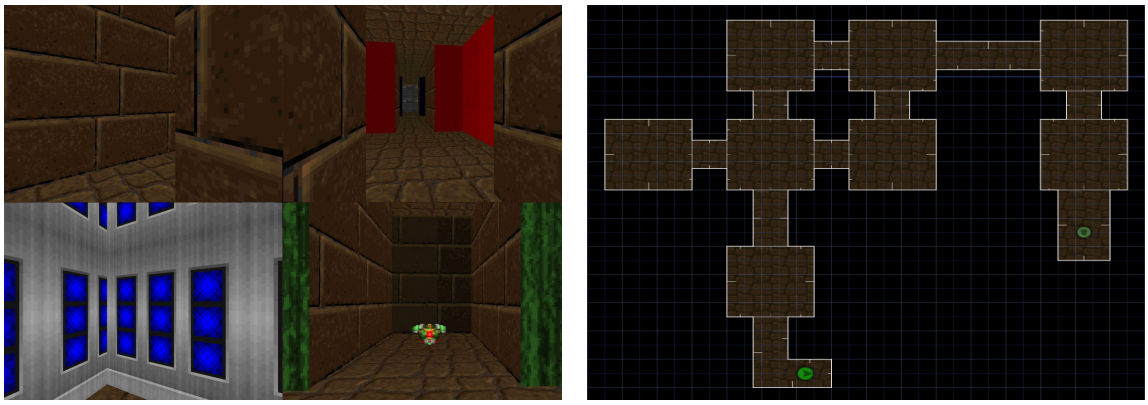
Unfortunately, there exist distractors that have none of said three properties, and thus, they blend perfectly within any interesting environment. These pose an intriguing new problem since a detect-neglect strategy is no longer viable for dealing with them. And consequently, they can also be expected to push learners to explore their pointless state spaces in the same way as they would do with other relevant elements of an environment. For that reason, here they are named exploration-intensive distractors. The archetypal example is a Rubik's cube, which is controllable, deterministic (though exploration-intensive distractors can exhibit stochastic dynamics as well), and is known to have pairs of configurations such that the minimum number of moves needed to transition from one to the other is 20. Moreover, its state space being virtually infinite is the cherry on top that makes this distractor overly perplexing, as that may sway some learners to play endlessly with the cube.

5.2 Contributions of this Benchmarking

The current literature reveals no major studies focusing on exploration-intensive distractors, and neither is there a well-known benchmark environment showcasing them. In light of this, as first contribution, two new environments displaying this class of distractors have been developed in this work. While a second contribution is a benchmarking covering these two environments and three accomplished algorithms, namely RND [12], ICM [74] and ECO [84]. Results from this benchmarking demonstrate that none of these methods manages to solve both scenarios and, more importantly, they do not exhibit an exploration policy fitting for this kind of domain. Meaning that, these exploration-intensive distractors represent an open question for which new interesting research is needed. The third and last contribution is the discussion of possible solution alternatives to this problem. Here, a particularly intriguing one is the deployment of a set of policies that could self-adapt in such a way that each ignores a different dynamic element of the world.

5.3 Related Work

In [55], the authors present an environment comprising a robotic arm and two balls. One ball can be grasped by the manipulator, while the other takes the role of a distractor. The arm can see this second ball moving around in a random walk, but it cannot touch it. Essentially, it acts as a phantom object. In [49, 96], pixel-level white noise is applied to various environments. It is added probabilistically both to chosen image observations and to chosen regions within those images. The agent has no control over this noise; it cannot influence its magnitude or its occurrence. In [84], a 3D maze is augmented with a broken TV that covers a quarter of the agent's view at all times. The TV shows only static, whose pattern invariably changes at each time step. In [11], a noisy TV is also introduced into a 3D maze. In this case, the agent is endowed with a special action that allows it to change the TV station. Importantly, every next station is drawn at random. A similar implementation is found in [84].



(a) (b)

Figure 5.1. Miscellaneous views of MyWayHome. Green circles in the layout indicate start (left) and goal (right) locations.

5.4 Environment Design

This section provides detailed specifications of two proposed sparse-reward environments, which showcase exploration-intensive distractors fashioned as TVs.

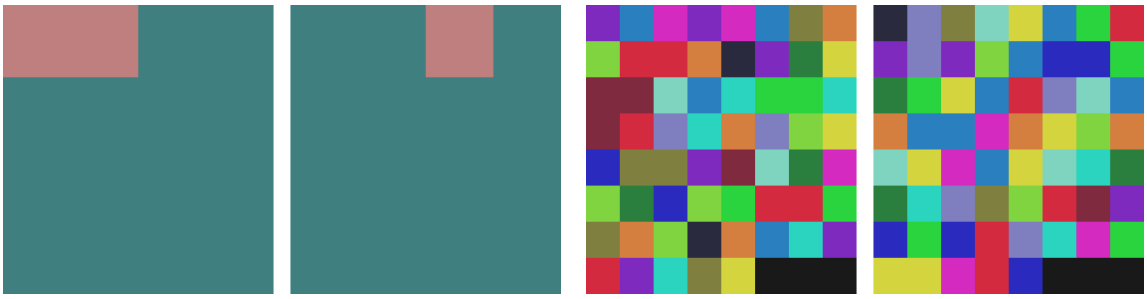
5.4.1 Base Environment

Following in the footsteps of [74, 84], these environments too are built upon the very sparse version of the MyWayHome task of ViZDoom [48]. This a visual navigation task that compels an agent to traverse a maze of corridors and rooms of varying textures (see Fig. 5.1). The agent's start location remains constant across episodes, though its orientation varies randomly between them. Its goal is to get a green armor, found always furthest from its initial location. Reaching this armor is a terminal event that yields the only positive reward of the environment.

In terms of observations, this environment generates first-person view images of the world, whose resolution and color format can be specified by the researcher. Concerning resolution, there are plenty of options ranging from 160x120 to 1920x1080. As for color format, it is possible to choose between RGB, RGBA, grayscale and Doom's 2 original palette. With regard to the agent's action set, ViZDoom offers up to 43 actions. Nevertheless, this set has been constrained to just three actions: turn right, turn left and move forward; in accordance with [84].

5.4.2 Single-Action Static-Rendering (SASR) TV

The first TV distractor allows an agent to change the station with just one action. The internal state of this TV is an L-digit base-b number, where L and b are set by the researcher. In particular, to define L, one must first configure the desired number of rows (r) and columns (c) into which the TV is to be partitioned; then $L = r \cdot c$. Nonetheless, all experiments conducted here have used $p = 2$ and $r = c = 4$ for this distractor. In other words, its internal state is a 16-digit binary number.



(a) (b)

Figure 5.2. (a) State transition from 3 (left) to 4 (right) in SASR TV. (b) State transition from 0 (left) to 1 (right) in MADR TV.

Such a state is set to zero at the beginning of every episode. Then, it is updated by incrementing it by one whenever the agent requests to change the station.

Eventually, this internal state is converted into an image observation that can be perceived by the agent (see Fig. 5.2(a)). Such rendering is accomplished in a straightforward manner, with the help of a constant colormap that assigns a unique color to each symbol used in base-b. This means that the same symbol always encodes the same color. Note that the color format of this TV is another configurable feature that can be set to either RGB or grayscale. To summarize, a TV station is visualized as an r by c grid. There, each cell is linked to a specific digit of the internal state and the cell's color is determined by the current symbol (value) of that digit via the constant colormap.

5.4.3 Multi-Action Dynamic-Rendering (MADR) TV

The construction of this second TV distractor follows many of the same considerations as the previous one. Just as before, the researcher is free to pick the color format, as well as parameters b, r and c. Likewise, these parameters shape the representation of the TV's internal state. In this case, such arguments have been set to $b = 16$ and $r = c = 8$ in all related experiments. This gives a 64-digit hexadecimal number as the internal state, which again starts at zero in every episode.

However, this distractor has three key differences. First, an agent is endowed with four actions for switching the station. Each action updates the internal state by adding or subtracting b^E from it, where E is an integer exponent initialized to zero at every episode. Having multiple actions grants the agent the chance to travel anywhere in the vast space of the TV; something impossible with a single action. The exact action set and corresponding transitions are as follows (the notation $\langle$internal state-exponent$\rangle$ is used in the examples):

- Station up: Increases the internal state by b^E. E.g., $\langle 36_{16}, 1_5\rangle \rightarrow \langle 46_{16}, 1_5\rangle$
- Station down: Decreases the internal state by b^E. E.g., $\langle 36_{16}, 1_5\rangle \rightarrow \langle 26_{16}, 1_5\rangle$
- Exponent up: Increments E by one, then increases the internal state by b^E. E.g., $\langle 36_{16}, 1_5\rangle \rightarrow \langle 136_{16}, 2_5\rangle$

- Exponent down: Decreases the internal state by b^E, then decrements E by one. E.g., $\langle 36_{16}, 1_5 \rangle \rightarrow \langle 26_{16}, 0_5 \rangle$

The second difference refers to the rendering of the exponent along with the station. Since the exponent plays a vital role in the TV's dynamics, the agent's observations are augmented with it to ensure that this distractor is predictable. With this intent, the internal state is split into two separate fields, one for the station and one for the exponent. Specifically, the size of the exponent field is given by the minimum number of digits needed to encode the size of the station field in base-5 (arbitrary choice). In this way, a 64-digit internal state comprises 3 exponent digits and 61 station digits. The rendering of exponent digits always occupies the bottom right corner of the TV grid. Moreover, it is only available as grayscale and it relies simply on a constant colormap for its implementation.

The third adjustment involves the dynamic rendering of station digits. Here, by dynamic it is implied that a symbol in the internal state can be translated into any out of b colors depending on the present value of the state. In detail, first, a constant matrix of size $b \times M$ ($M = 10^6$) is defined, whose elements are integers uniformly distributed between 0 and $b - 1$. Next, a symbol $s \in [0, b - 1]$ is mapped to a color by following 3 steps. 1) A count is obtained of the number of appearances of that symbol in all base-b numbers between zero and the state's value (station digits only). 2) From the former matrix, the element at row index s and column index equal to the modulo of the previous count with respect to M is extracted. 3) This element passes through a constant colormap to get the final rendering color. Fig. 5.2(b) depicts a transition in this TV. The main advantage of such dynamic rendering is that it virtually removes the possibility of generalizing the prediction of the forward dynamics from an understanding of basic arithmetic. A prospect that could conceivably lessen the curiosity of an agent towards this distractor.

5.4.4 ViZDoom-TV Integration

As a final step, the MyWayHome task is integrated with the previous TV distractors. Unlike [84], where an agent simultaneously observes the maze and the TV at all times, in this study the agent is granted an extra action that switches between the maze and TV full-screens. Therefore, the tasks resulting from consolidating the SASR and MADR TV distractors into MyWayHome have a total of 5 and 8 actions, respectively. A detail worth mentioning is that TV actions are inconsequential in the maze view (stations do not change), whereas navigation actions maintain their expected effects in the TV view (the agent moves regardless).

Having a view-switching action takes away any limitation on the relative size of the TV. In fact, with the help of a black background, the area containing meaningful information is made four times larger in the TV view than in the maze view (see Fig. 5.3). Overall, magnifying the size of the distractor is anticipated to make the problem harder as it makes it less likely to be overlooked.

Figure 5.3. Side-by-side of maze and TV views.

5.5 Benchmarking Setup

This section recounts every configuration detail regarding the battery of experiments conducted within the two previous environments as part of the present benchmarking undertaking.

5.5.1 Covered Algorithms

The algorithms taking part in this benchmarking are the three detailed in Subsection 3.3, namely: RND, ICM and ECO. A brief a priori theoretical analysis on how these methods would react to the two distractors proposed here on the grounds of their formulations reveals the following. Since RND assigns larger bonuses to more novel observations (i.e., never-before-seen observations), it should struggle with a distractor that can assume nearly infinite unique and easily reachable states. This distractor permits the existence of policies that visit a new state on each time step of every episode, which would naturally collect more rewards than policies trying to traverse the same maze over and over again. Ergo, RND will likely prefer the former ones with so much intent that it could become completely fixated on them. Examples of distractors of this kind include: pixel-level noise, a broken/noisy TV, a Rubik's cube and the MADR TV designed in this benchmarking. By construction, thanks to having an inverse dynamics model that grounds its internal representation, ICM tends to be resilient against uncontrollable distractors (e.g., phantom objects, pixel-level noise). However, because its bonuses are correlated to the prediction error of its forward dynamics model, it should fall victim to distractors that though compliant are highly stochastic (e.g., a noisy TV) or chaotic. Notably, the MADR TV falls into this later category, since while being deterministic similar sequences of actions will likely lead to disparate series of TV stations. Lastly, ECO has been designed to ignore any distractor that demands a low transitioning effort; such as: pixel-level noise or a broken/noisy TV. Therefore, by definition, it is expected to be affected by the SASR and MADR TVs, which has been engineered precisely to demand a great exploration effort.

5.5.2 Training Details

In terms of preprocessing, a frame skip of 4 steps is enforced in every tested environment, where only the last frame is passed through without any max pooling. Additionally, a timeout of 2100 steps is set at the engine's level, which translates to 525 steps at the agent's level (because of the frame skip). Moreover, all three algorithms have been run using observation formats comparable to the ones found in their original papers. Hence, for RND and ICM, each observation is put together as a stack of 4 consecutive 84x84 grayscale game frames. Meanwhile, ECO receives as input a single 84x84 RGB image.

Regarding neural architectures, an identical multi-head policy-value network is employed by all methods. Such a network consists first of a stack of 4 convolutional layers with 32 filters each, kernel size of 3, stride of 2 and same padding. Then, the output of this convolutional module is flattened and passed through 2 dense hidden layers with 2048 units each, before going to the output policy and value heads. RELU activation is employed in every convolutional and hidden layer, while linear activation is applied to all outputs. RND's random and predictor networks each use a convolutional module with the aforementioned topology, but instead their activations are ELU and leaky RELU, respectively. Both modules are fed only the last frame of each observation as input. The random network's module is then connected to an output linear layer with a representation size of 256. In turn, the output of the predictor's module goes through 2 dense hidden layers with 512 units each and RELU activation. ICM also adopts the previous 4-layer convolutional assembly for the encoder of its curiosity module, except that it replaces the activations with ELU. Its forward and inverse models are each built with a single dense hidden layer containing 1024 RELU units; plus, the former model has a linear output layer whose dimensionality is 1152. For its reachability classifier, ECO builds a siamese network where each of its two branches has a replica of the convolutional module used by its policy. These branches are followed by a stack of 4 dense hidden layers with 512 RELU units each and, subsequently, by a softmax layer with 2 neurons.

Table 5.1 gathers the remaining hyperparameters used in this benchmarking.

5.6 Results

RND, ICM and ECO were evaluated three-ways on the very sparse MyWayHome environment: without any TV (baseline), with a SASR TV and with a MADR TV. In all cases, a black background covered 75% of each ViZDoom frame, as explained in Section 5.4.4. The baseline experiments were stopped after 10 million frames, while the others went on for 10 million more to account for the added difficulty of facing distractors. For reference, regardless of the algorithm, a single run of 20 million frames takes about 1 to 2 days to complete using an NVIDIA A100 GPU together with an 8-core 16-thread processor. Each experiment was repeated with three different seeds. The following graphs report simple averages across runs that, except the historical station count, are derived from cumulative averages taken along each run.

Table 5.1. Overview of applied hyperparameters

Hyperparameter	RND	ICM	ECO
Number of workers	32	16	16
Rollout length	128	80	256
Number of optimization epochs	4	4	4
Learning rate	1.0e−4	2.5e−3	2.5e−4
Learning rate decay	None	Linear	Linear
Entropy coefficient	0.01	0.01	0.01
PPO (λ)	1.0	1.0	0.95
Coefficient of extrinsic rewards	500	1.0	1.0
Coefficient of intrinsic rewards	1	0.01	0.03
Extrinsic discount factor (γ_E)	0.99	0.99	0.99
Intrinsic discount factor (γ_I)	0.99	-	-
RND experience fraction used by the predictor	1.0	-	-
ICM curiosity loss strength (λ)	-	0.1	-
ICM forward inverse ratio (β)	-	0.2	-
ECO episodic memory size	-	-	200
ECO action distance threshold	-	-	5
ECO negative sample multiplier	-	-	5
ECO reward shift (β)	-	-	0.5
ECO novelty threshold ($b_{novelty}$)	-	-	0.5
ECO aggregation function (F)	-	-	P_{90}
ECO reachability training learning rate	-	-	1.0e−4
ECO reachability training history size	-	-	16K

Baseline experiments: Fig. 5.4 demonstrates that all three approaches are capable of solving the baseline task. In particular, every solution converges to a policy that on average travels 14 sectors per episode (rooms & corridors), which is the minimum necessary to reach the goal (as seen in Fig. 5.1).

SASR experiments: Only ICM manages to solve this problem satisfactorily on most occasions (see Fig. 5.5). This result is not unexpected given that in practice this TV distractor conveys an ordered sequence of just 525 predictable transitions. With such a few TV transitions, an ICM agent is bound to experience them more frequently than those relative to the maze, especially the ones closer to the initial station. Consequently, its curiosity remains constantly lower for the TV than for the maze (see Fig. 5.7), which pushes the agent to prioritize the exploration of the latter over the former. The same line of reasoning applies to an RND learner. It should quickly get bored with a TV that has few stations (merely 525) and switch its attention to the maze. However, surprisingly, this is not what happens. Instead, through a deeper inspection, it has been identified that its policy swiftly converges to a uniform distribution across the entire action set. As a result, what the graphs expose is an agent that keeps the TV on exactly half of the time and makes no significant exploration of neither the maze nor the TV. On the other hand, ECO is

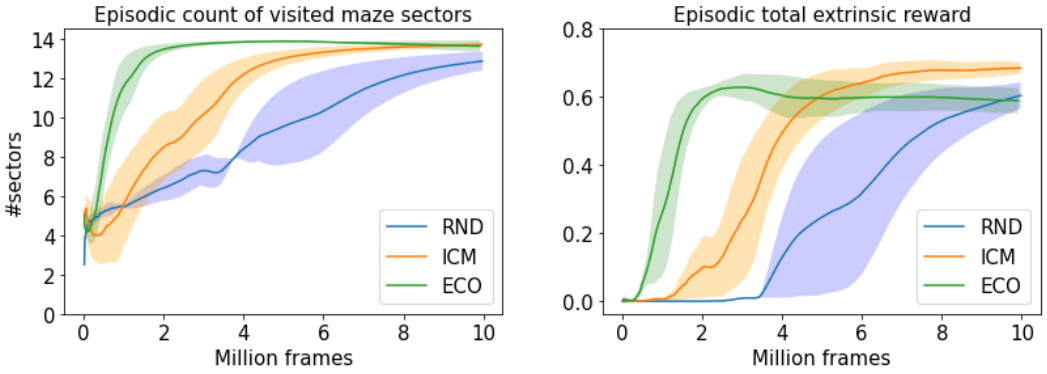


Figure 5.4. Baseline results.

the single method that becomes obsessed with this TV. Running its final policies on testing mode reveals that ECO learns the trivial strategy of always changing the station. A policy that is optimal for collecting intrinsic rewards since it guarantees maximizing the action distance between final and initial stations within an episode. As to why this method completely ignores the maze; it is presumably because of the low complexity of the SASR TV, which makes it possible to learn an effective far-reaching policy much faster in the TV than in the maze. Once found, such policy denotes a global maximum of the intrinsic return and, as such, seriously hinders the exploration of other parts of the environment.

MADR experiments: As shown in Fig. 5.6, none of the three benchmarked methods solves this environment. RND and ICM consistently get fairly fixated on this TV, i.e., the fraction of time they have the TV on is well above 0.8. Furthermore, they essentially do not explore the maze seeing that they never visit more than about 2 sectors on average per episode. Looking also into the historical number of stations visited by these two algorithms, this count grows linearly, which is evidence that they continuously discover new stations and transitions throughout a run. A behavior that is perfectly aligned with their theoretical motivations for novelty and curiosity. To figure out why this happens concretely, the final policies of RND and ICM are run on testing mode (frozen policies) for a few extra million frames and each time the historical count of visited stations is collected. These additional results indicate that for both approaches the slopes during training and during testing of such a count are almost identical to each other. This suggests that these methods learn high-entropy stochastic policies that work everywhere in the MADR TV. Such solutions are additionally rather trivial given the extremely large state space of this distractor and its chaotic dynamics, i.e., two very similar sequences of actions will produce two greatly different sequences of states. Indeed, it is easy to see that because of those qualities, even a random policy over the 4 TV actions will reproduce the previous slopes, since during an entire run it will hardly perceive twice states or transitions occurring a couple dozen actions after the beginning of an episode. Putting all these findings together, the learning process of RND and ICM arguably goes roughly as follows. They quickly learn such primarily random policies; as seen in the historical station count graph where their linear tendency is visible from the very

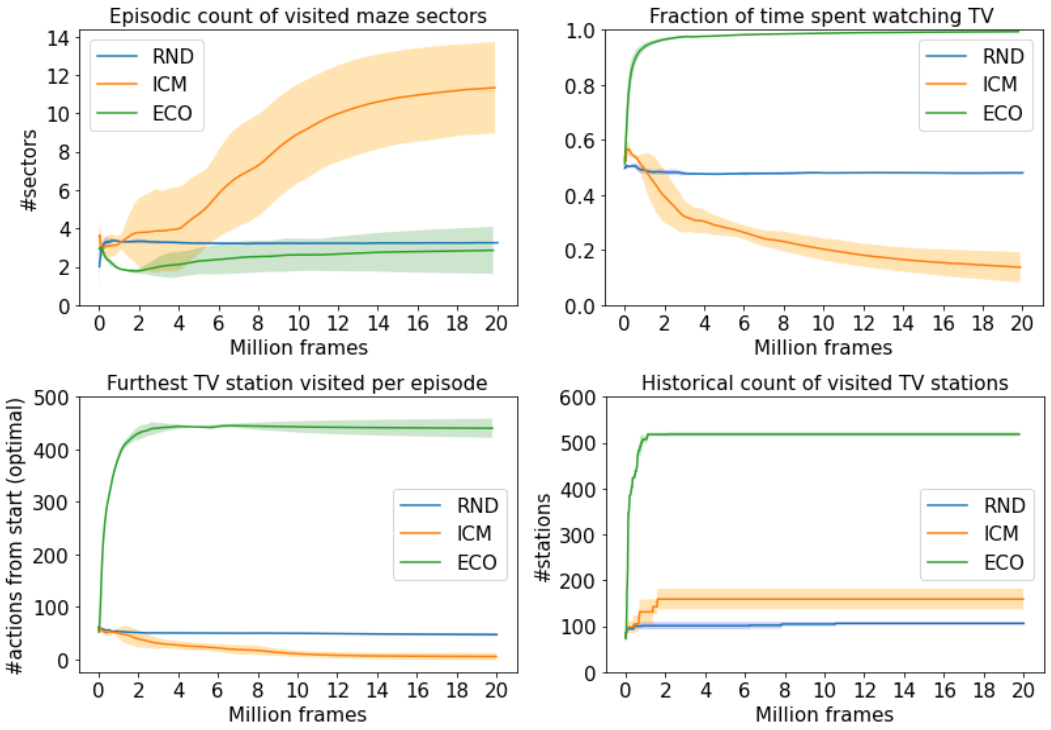


Figure 5.5. SASR results.

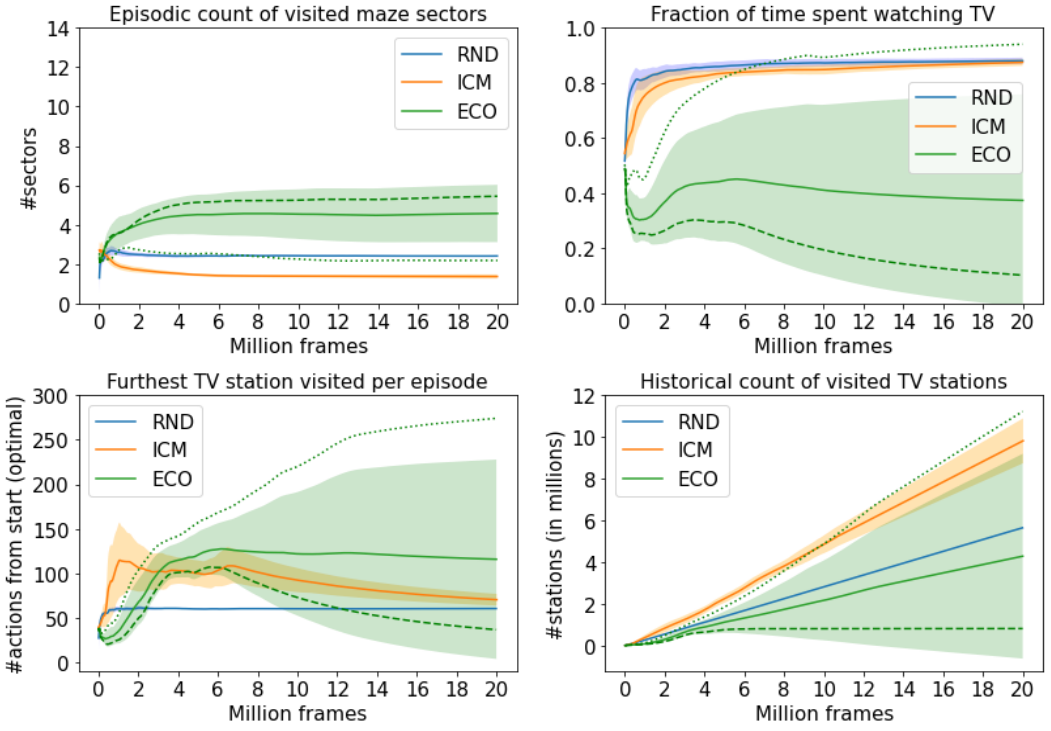


Figure 5.6. MADR results. Just for ECO, these results are further disaggregated into two averages condensing the runs where the learner gets captivated (dotted lines) or not (dashed lines) by the TV.

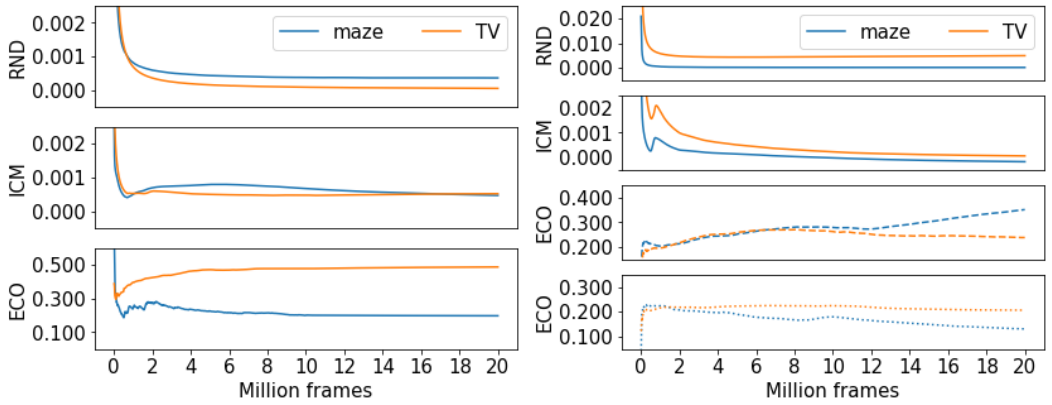


Figure 5.7. Per-step intrinsic reward disaggregated between maze and TV states occurring in the SASR (left) and MADR (right) experiments. Dotted and dashed lines denote the same decomposition as in Fig. 5.6.

beginning. These policies gather a meaningful episodic intrinsic return that does not wither down significantly over time considering they mostly observe novel states and transitions from start to finish of the training. Ultimately, those policies constitute hard and permanent local minima of the PPO optimization objective, as getting a return of the same order of magnitude while exploring the maze requires vastly more effort and experiences. Moving on to ECO, noticeably, its learning evolution varies drastically across runs. In one run, its full attention is drawn to the TV as it also exhibits a will to reach increasingly remote stations, settling in at an average distance of over 250 actions from the start. Whereas, in the other two runs, it steadily loses interest in the TV; eventually having it off most of the time. To better understand this contradiction, an examination of the accuracy of the reachability networks emerging during these runs was undertaken. This reports that ECO consistently (in all 3 runs) converges to an erroneous generalization regarding the reachability landscape arising from the MADR TV. One in which two TV stations are close to or far from each other depending on whether the difference between their exponents is respectively below or above a threshold of around 5 units. This mistake is not truly shocking; two stations many exponents apart will always be distant from each other and this concept is surely faster to grasp than the true action distance function. Still, it is relevant to wonder why ECO never evolves beyond this error. This is quite possibly due to the astronomical number of stations and the even larger number of pairings between them originating from this TV, as well as the lack of helpful clues for the generalization of distances contained in individual observations. In the end, it seems unrealistic that any practical neural architecture would have enough capacity and would see enough examples to learn that much information. In any case, to maximize the intrinsic return stemming from this TV under the previous faulty reachability model, this examination further exposes that ECO always acquires one simple but effective policy. It almost exclusively enacts either exponent up or exponent down. However, since such strategy stops seeing hard-to-reach states after the first 60 steps of an episode (this TV has only 61 exponents), its associated intrinsic return ends up being merely a local maximum. It is then here hypothesized

that this is the primary reason leading to the disparate results shown by ECO. On some runs, due to their random seeds, the learner finds the local maximum lurking in the TV so fast that it thereafter diverts all attention away from any other less rewarded search effort. Other times, the maze and the TV are explored more in unison until the return associated with the latter hits its limit. From that moment onward, the agent receives higher returns by focusing purely on the maze, prompting its progressive neglect of the TV.

5.7 Discussion

Some final remarks before closing this section. This benchmarking reveals that ICM and ECO, two algorithms designed to handle distractors by ignoring them, do not fundamentally dismiss both proposed exploration-intensive distractors, which validates their research relevancy. Furthermore, it is a little unsatisfying that a solo distractor is not able to impair all 3 benchmarked approaches. Nonetheless, to make this happen, a simple idea is to merge these two distractors; for instance, by allowing the view-switching action to cycle between the maze, the SASR TV and the MADR TV. In such an environment, every algorithm that has been hypnotized by a distractor in this benchmarking should succumb to the exact same fate. Lastly, one pending concern must be answered. Is there a principled way to deal with exploration-intensive distractors? In fact, it is reasonable to envision a learner that through experience becomes aware of which elements with recognizable local dynamics exist in its world, e.g., a TV or a Rubik's cube. Having identified these, a sensible solution is to concurrently learn an array of exploration policies where each one focuses on an imaginary world in which a particular set of dynamic elements has been abstracted away. For example, with reference to herein analyzed TV distractors, three policies are needed: one that ignores the TV, one that ignores the maze and another that ignores neither. Provided the learner also guarantees allocating sufficient exploration resources to each policy, then with certainty at least one will crack the problem while paying no attention to the distractor. Needless to say, none of the algorithms included in this benchmarking displays this insight.

Chapter 6

Benchmarking Re-Exploration Demanding Environments

Sparse-reward environments are notoriously difficult for deep reinforcement learning (DRL). Yet, the prospect of solving them without relying on reward engineering is highly appealing. So much so that in the last few years numerous DRL algorithms have been specifically designed to tackle sparse rewards automatically. In this chapter, we test the competence of these methods in a type of common sparse-reward problem that has been largely ignored so far. Namely, our focus is on sparse tasks that demand the re-exploration of an environment within the present episode. To this end, we first develop two novel domains displaying the former need, each posing distinct challenges. Then, we conduct a benchmarking in these domains involving three state-of-the-art algorithms of proven success at handling sparse rewards. The results of this study clearly indicate that current methods are not at all suited to deal with re-exploration. Nevertheless, we do not see any fundamental reason as to why problems of this kind should remain intractable. Thus, we expect this work, and especially the benchmarks we are making publicly available, to promote further research in this engaging topic.

6.1 The Critical Role of Memory Within Exploration

Learning with sparse rewards is an stimulating research area that have seen many past failures as well as many recent successes, and one in which to date there still remain many unresolved issues. In this work, we expose and analyze one of these; specifically, our focus is on sparse-reward environments where an agent is forced to re-explore the world before reaching its goal. Such domains are commonplace in video games as well as in everyday life. The quintessential example is that of an agent that although being in front of a locked door, to open it, it must first search elsewhere for the corresponding key. Critically, the agent has to navigate extensively before finding said key, which is situated somewhere at the middle the world. And after that, as it makes its way back to the locked door, it has no choice but to re-visit the same locations seen along the path leading to the key. Notably, the former scenario is only problematic for DRL techniques when the agent possesses partial observability of the domain. In particular, when it is unable to directly perceive the

current state of the key (collected or not) from most locations. Under such condition, we are not aware of any algorithm, including every approach cited in the previous paragraph, that has the theoretical basis for solving the previous example. This is because the exploration strategies applied by all these methods lack a sophisticated sense of episodic memory. In other words, they are incapable of understanding that identical observations before and after collecting the key may actually correspond to two very different states. Such a claim alludes obviously to techniques such as RND whose intrinsic motivation is oblivious to past events within an episode; that is, their exploration apparatus processes just the current observation or transition. But it also applies to algorithms akin to Agent57, which possess some rudimentary form of episodic memory. Although such a gadget allows them to detect if an observation has been spotted before along the ongoing trajectory, they still fail to acknowledge it as related to a brand new state when such is the case. Ultimately, referring to our initial example, this across-the-board confusion pushes both kinds of methods to regard places visited before getting hold of the key as too boring and not worthy of being re-explored during the remainder of the current episode. Preferring instead to survey the not-yet-seen parts of the environment that lie beyond the key and away from the locked door.

6.2 Contributions of this Benchmarking

In our review of the literature, we have not encountered studies paying attention to the problem of re-exploration as it has been described here, and neither have we come upon proper benchmark environments highlighting this issue. Therefore, as first contribution, we have developed three new domains whose resolution demands re-exploration, and which we have made open to the community. They all embrace the locked door & key paradigm as main task, but each features uniquely interesting complexities. Our second contribution is a benchmarking involving these three environments and three influential algorithms; namely, RND [12], ICM [74] and ECO [84]. This experimental study unequivocally reveals that these methods never manifest any sort of re-exploration behavior. A result that strengthens our previous claim that they are simply not equipped with the necessary mechanisms for such feat. However, as we take notice of the advanced usage of memory execute by humans and animals, it is hard to label re-exploration as an unapproachable endeavor. Instead, we consider it an issue soon to be resolve, provided a bit more research effort is put into it. In that sense, as a final contribution, we discuss a few possibilities of what could be done to endow the algorithms partaking in this benchmarking with the ability to re-explore their world.

6.3 Related Work

DRL researchers have followed several miscellaneous avenues to find solutions to the problem of sparse rewards. We have compiled several of them in Section 3.2, but next we are providing a deeper explanation of individual algorithms and putting special attention to their capacity to remember within the scope of exploration. Various interestingness measures such as novelty, curiosity, empowerment, surprise,

compressiveness and many others have been proposed to provide intrinsic motivation. Novelty-based exploration guides agents to states that have been visited less often. For example, [9] relies on a trainable memoryless density model to compute novelty in terms of pseudo counts. Likewise, Agent57 [76] takes advantage of novelty, which in this case is separately computed at two levels: within each episode and across an entire training session. In the end, it is the multiplication of these two quantities that gets maximized in the absence of extrinsic reinforcement. Importantly, such an episodic novelty is calculated as some manually-crafted similarity between the latest observation and all previous observations seen during the ongoing episode, which are stored in a resettable memory buffer. In turn, curiosity-motivated learners will prefer to visit areas of the world where transition dynamics is not yet well understood. In [97], curiosity is defined as the error shown by a feed-forward neural autoencoder trained to predict one-step forward dynamics. With respect to empowerment, an agent maximizing this measure will be driven towards states from which more future states are reachable. For instance, [66] learns a tractable mutual information lower bound decomposed over multiple models, one of them being directly an empowerment estimator implemented as a multi-layer perceptron fed with just the current state. A further approach for tackling sparse-reward environments embraces the idea of learning one or more convergent policies that cover the entire state space within a single or few episodes. This exact idea is enacted in [57], where the entropy of an approximate marginal distribution over individual states is adopted as a maximization objective for a set of memoryless exploration policies. Another compelling line of research uses approximations to Bayesian uncertainty as guidance for deep exploration. In [72], while training a multi-head non-recurrent Q-network, in the spirit of Thompson sampling, exploration is enforced by randomly selecting a head at the beginning of each episode and following its induced policy thereafter. Alternatively, grounded on the formulation of an additional Bellman equation, [69] builds a feed-forward Q-network that concurrently estimates mean Q-values (as usual) and their variances. This information is then sent to a selection rule designed to prefer actions leading to higher returns with higher uncertainties. In [90], Bayesian uncertainty is directly regarded as intrinsic rewards; here, it is approximated as the ensemble disagreement from 5 multi-layer perceptrons trained in parallel to predict one-step transitions. Promoting diversity among a large number of skills has also been demonstrated to help with sparse rewards. As an example, [23] updates multiple skills simultaneously based on pseudo-rewards that incentivized them to visit non-overlapping regions of the state space. Such rewards are supplied by a non-recurrent discriminator trained to distinguish skills given just the current state. In multi-goal scenarios, [2] reveals great benefits against reward sparsity by leveraging hindsight and goal re-labeling to learn from mistakes. However, methods following this notion require extra knowledge of a goal space, and if this is not available the most general alternative is to substitute it with the observation space. Yet another behavioral strategy useful in scenarios where rewards are rare is to imitate a learner's own successful past experiences. Case in point, [70] defines a self-imitation loss applied independently to each stored one-step transition that compels a policy to repeat past actions that reported a cumulative reward larger than the current value estimate. Automatic curriculum learning has been shown to progressively steer exploration towards regions further and further away from the initial state

in environments devoid of rewards. For instance, in [28], exploration results from commanding a goal-conditioned policy to always visit states of intermediate difficulty. These are generated by a GAN that, given previously seen states and accuracies, produces a distribution of imagined states with the right probability of being reached by the current policy. In [29], the resettability of the environment is exploited to create a reversed curriculum that generates a progression of initial states that moves away from at least one known goal state. In [100], an exploration curriculum arises from two learners playing a game. One is rewarded for posing challenging goals, represented as states (or observations in partially observable environments), while the other is rewarded for reaching them.

From this small yet representative summary, one constant becomes clear among almost every referenced approach and algorithm. Their exploration machinery acts exclusively based on individual states (and sometimes also actions), supported purely by memoryless auxiliary models, e.g. feed-forward neural networks. To make things worse, in partially observable environments, it is common practice to simply replace individual states with individual observations. Therefore, it is hard to conceive how methods that lack a memory mechanism and do not see beyond the current observation could discriminate between different world states conflating to the same observation. An ability we believe to be critical for triggering re-exploration, and thus, tackling the benchmarks proposed in this work. There is just one exception within this sample of techniques in terms of the ability to remember, Agent57 indeed keeps an episodic memory buffer. Nonetheless, such a component is merely useful for determining if an observation has been seen multiple times in a single episode. Yet, as it has been implemented, it does not provide any insight about the hidden world state via novelty bonuses. To illustrate this, recall the example environment presented in Section 6.1. There, a given location will report fairly similar novelty measures according to Agent57, whether the interacting agent re-visits it after collecting the key or it follows almost the same round-trip trajectory but only comes very near to the key without actually grabbing it.

6.4 Environment Design

We propose two new benchmark environments featuring the need for re-exploration. As previously mentioned, they all implement the locked door & key paradigm presented in Section 6.1. Furthermore, these environments are identical in every way and only differ from each other in one key property: the absence or presence of deadly states.

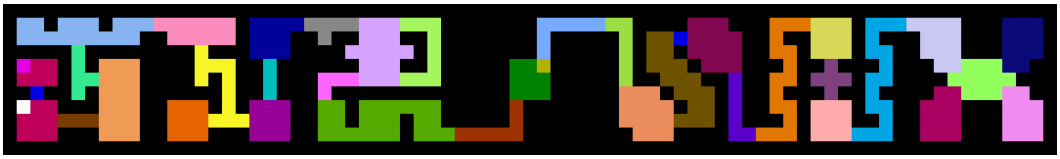
We will begin by describing our harmless domain. As depicted in Figure 6.1(a), this task transpires in a 7-room gridworld maze, where each room is made of 11x11 cells. The avatar's goal is to navigate its way through this maze and eventually collect the jewel found in the leftmost room behind a locked door. Importantly, to unlock the latter, this agent needs to first grab the key found in the fourth room, exactly at the middle of the maze. Going into the technicalities, the interacting agent starts every episode in the same cell; one cell down and one to the left of the locked door located in the first room. Hence, in every simulation, to solve the task, the avatar must re-visit the three leftmost rooms. In terms of perception, the agent

understands the world through images. Additionally, it has partial observability. Meaning that, at each time step, the only true information it receives about the present state of the environment comes from an image of only the room it currently is in. Aside from that, we left the precise setup of the observation space up to each researcher, which are free to try various configuration options regarding color format, resolution and frame stacking. With respect to its action space, this domain considers 4 discrete actions denoting four directions of motion: right, left, down and up. If an action would make the avatar go through a wall, it will not be executed since walls are impassable; the agent will remain in its current cell. It is worth mentioning that there are no special actions for grabbing the key or the jewel, or for opening a door. These outcomes happen automatically after stepping over the cells containing such items. Another critical dynamic is that when the key is taken, it disappears entirely form the screen, and stays that way for the remainder of the ongoing episode. This ensures that at least for the three leftmost rooms, the agent has no direct means of knowing the state of the key. To make this domain more stimulating, we have also introduce stochasticity by implementing sticky actions, such that with probability p the latest action selected by the agent is superseded by the one taken in the previous step. Concerning its reward structure, this environment hands one non-zero incentive of $+1$ when the jewel is collected. On every other occasion the reward given to the agent is zero. Lastly, an episode terminates under just two conditions: precisely when the jewel is taken and when a preset timeout of 320 interaction steps expires. Importantly, this timeout has been specifically tuned, such that it is sufficient for the agent to get the key and then the jewel, but it is impossible to visit all 7 rooms forward and backward. The latter is an uninteresting yet effective strategy that some algorithms are capable of developing even while lacking a sophisticated understanding of an episodic memory. Thus, we urge not to increase the value of this timeout while doing tests in this environment.

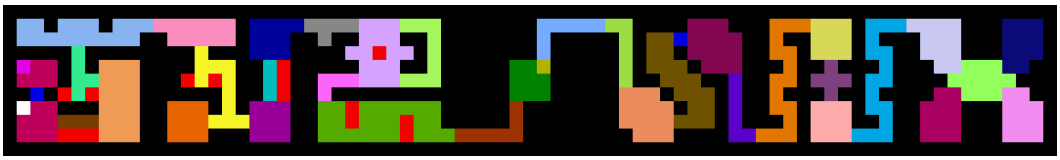
Referring now to the deadly environment we are further proposing. As seen in Figure 6.1(b), this happens in a maze with an almost identical layout as the previous one. Not only that but every single operational detail reported above also applies to this second domain. The sole difference is that this second maze is peppered with plenty of deadly cells, found exclusively in its three leftmost rooms. As their name suggest, the agent is immediately killed if it lands on any of such cells, i.e., the episode ends. In other words, these cells add a third termination condition. Crucially, we have programmed such lethal cells to be active only after the key is grabbed by the agent. Before that, they are actually harmless (in fact they function as a wall). Notably, their visibility remains unchanged whether being active or inactive. With these deadly cells, our primary intention is to give extra discouragement to any method attempting to retrace its steps within the span of an episode.

6.5 Benchmarking Setup

In this section we present a benchmarking study, which confronts the previous two environments with three accomplished DRL algorithms proven to handle sparse-rewards.



(a) Harmless scenario.



(b) Deadly scenario.

Figure 6.1. Full views of all rooms at the initial states of our two maze environment. Color code: white (avatar), blue (door), magenta (jewel), golden (key), black (walls), red (deadly states); everything else are traversable sectors (34 in total).

6.5.1 Covered Algorithms

The three methods partaking in this benchmarking are: RND, ICM and ECO, which have already been detailed in Section 3.3. With regard to ECO, we obtain better results by making a minor modification of its bonus scheme. Namely, we dictate that a bonus of $+1$ is awarded whenever an observation enters its episodic memory, while bonus of zero is delivered in every other situation.

These three algorithms follow the patterns discussed in Section 6.3. Inside their exploration machinery, RND and ICM perceive exclusively immediate observations or transitions and do not possess components with any memory capacity. Thus, by definition, they will always assign an equal bonus to identical inputs, regardless of whether or not their underlying world states are organically different. Furthermore, as these two bonuses normally remain unchanged across an entire episode, most of the time the optimal strategy is to travel to the observation or transition denoting the highest intrinsic reward and stay there. All in all, it seems hardly conceivable beforehand that such bonuses could foster the re-exploration needed to solve our proposed domains. On the other hand, the episodic memory of ECO operates analogously to that of Agent57. It helps to detect if an observation has been sighted before, but still does not inform if the environment has been modified beyond the immediate perception. Even though this buffer may contain enough information to deduce the world state, ECO only uses it to compute its real-valued reachability bonus, a data compression that losses most of the knowledge existing within the episodic memory. It could be tempting to think that the ability of ECO to discriminate between first, second and more visits could drive it to retrace its steps. Certainly, in any of our custom-made environments, along a trajectory that goes to the key and then moves back, we will see that bonuses assigned to a particular observation are noticeably different before and after getting said item. The memory buffer is likely to store more observation the second time around, and that necessarily alters the bonus computation. Unfortunately, a priori, this is not expected to suffice for solving our tasks. To work properly, ECO and similar

methods must make sure that their bonuses decrease with re-visits. Otherwise, an optimal policy interested purely in maximizing bonuses could just iterate between two high rewarding observations. Ultimately, this means that bonuses received when backtracking from the key to the initial state are normally smaller than those gathered by traveling past the key and into uncharted territory. A bias that might just have the necessary strength to trump the possibility of re-exploration.

6.5.2 Training Details

In terms of preprocessing, in all cases, we configure observations to have a resolution of $44x44$ and we never apply frame skipping. For RND and ICM, we also consider frame stacking of 2 and grayscale as color format, which gives an observation shape of 44x44x2. But for ECO, we employ RGB images and no frame stacking, resulting in observations of size 44x44x3.

In reference to their neural models, we setup a multi-head policy-value network with virtually the same architecture across all three methods. Such network consists first of a stack of 4 convolutional layers with 32 filters each and kernel size of 3x3. In addition, the upper three layers have a stride of 2 and padding, whereas the last one shows a stride of 1 with no padding. Afterwards, before going to the policy and value heads, the output of this convolutional module is flattened and passed through a LSTM layer with 512 units while working with ICM and ECO, or through a GRU layer with 684 units in the case of RND. RELU activation is employed in every convolutional layer, while linear activation is applied in all output heads. Moreover, RND uses two convolutional modules of the same topology as before for its random and predictor networks, but instead of RELU their activations are leaky RELU. As inputs, we feed them merely the latest frame of each observation. The outputs of these two modules then go through 2 dense hidden layers (512 units/layer and RELU activation) and a linear output layer with a representation size of 256. In like manner, ICM adopts the previously described 4-layer convolutional assembly for the encoder of its curiosity module, except that it replaces the activations with ELU. Furthermore, its forward and inverse models are each built with a single dense hidden layer containing 512 RELU units; plus, the former model has a linear output layer whose dimensionality is 512 as well. Lastly, ECO articulates a siamese network as its reachability classifier, where each of its two branches has a replica of the convolutional module being used by its policy-value network. These branches are followed by a stack of 4 dense hidden layers with 512 RELU units each and, subsequently, by a softmax layer with 2 outputs.

Table 6.1 compiles all other hyperparameters used in this work.

6.6 Results

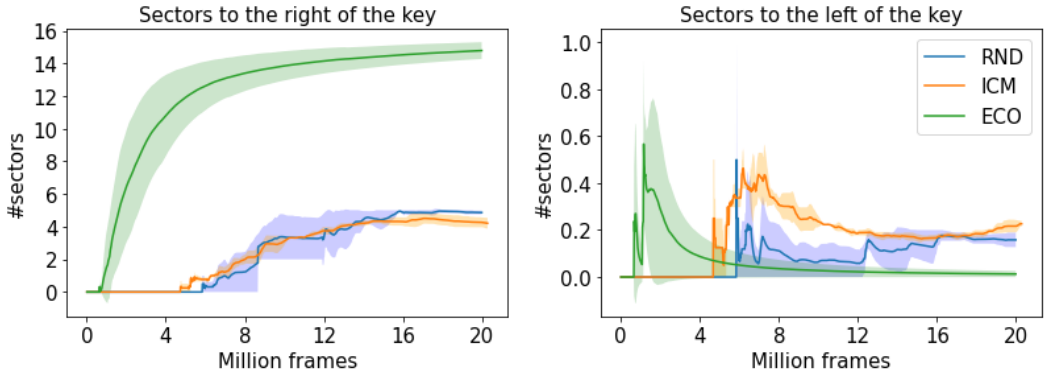
We have evaluated three algorithms: RND, ICM and ECO; on our two maze scenarios: harmless and deadly. All experiments (6 in total) were repeated with three different seeds and each run lasted a maximum of 20 million frames (interactions steps).

Figures 6.2 and 6.3 make it quite evident that the behaviors of the three benchmarked algorithms in our proposed domain align perfectly with our expectations (expressed in Subsection 6.5.1). Certainly, none of them showed any enthusiasm

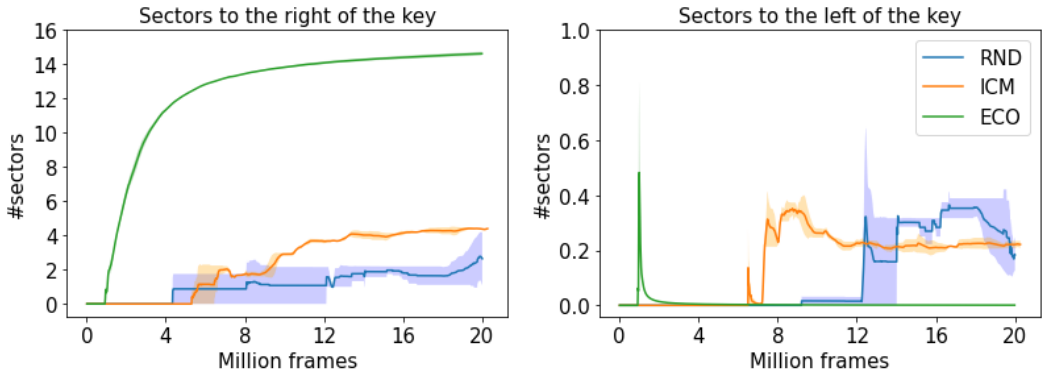
Table 6.1. List of hyperparameters

Hyperparameter	RND	ICM	ECO
Number of workers	32	16	16
Rollout length	128	80	256
Number of optimization epochs	4	4	4
Learning rate	1.0e−4	5.0e−5	2.5e−4
Learning rate decay	None	None	Linear
Entropy coefficient	1.0e−4	0.01	0.01
PPO (λ)	0.95	1.0	0.95
Coefficient of extrinsic rewards	2	1.0	1.0
Coefficient of intrinsic rewards	1	0.01	0.03
Extrinsic discount factor (γ_E)	0.999	0.99	0.99
Intrinsic discount factor (γ_I)	0.99	-	-
RND experience fraction used by the predictor	1.0	-	-
ICM curiosity loss strength (λ)	-	10	-
ICM forward inverse ratio (β)	-	0.005	-
ECO memory size	-	-	320
ECO action distance threshold	-	-	5
ECO negative sample multiplier	-	-	5
ECO reward shift (β)	-	-	0.0
ECO novelty threshold ($b_{novelty}$)	-	-	0.3
ECO aggregation function (F)	-	-	P_{90}
ECO reachability training learning rate	-	-	1.0e−3
ECO reachability training history size	-	-	128K

in re-exploring the environment. But what really surprised us was the magnitude of its disinterest in retracing their steps. These results indicate that, during an episode, after grabbing the key they were on average 20 times more likely to keep moving right (into new territory) than to move left (into already visited sectors). Not only that, but not once any of them went beyond the bottom green sector of the third leftmost room (i.e., the first sector when entering this room from the middle one). Meanwhile, in the other direction, all three algorithms reached the furthest right room at some point. Notably, since there was no meaningful re-exploration in any case, results relative to the harmless and deadly environments were essentially identical. With respect to RND and ICM, there is not much left to say. Both these methods lack any sense of memory whatsoever; thus, this empirical validation of them failing to backtrack seems fitting. On the other hand, the indifference of ECO toward re-exploration feels underwhelming, when considering that it has an episodic memory. A priori, we were not expecting it to complete the proposed domains, but we had expected it to reverse much more than what it eventually did. To understand why this was not the case, let's see how ECO distributes its bonuses. Recall from Section 6.5 that such bonuses are binary and that we are using a 90th percentile



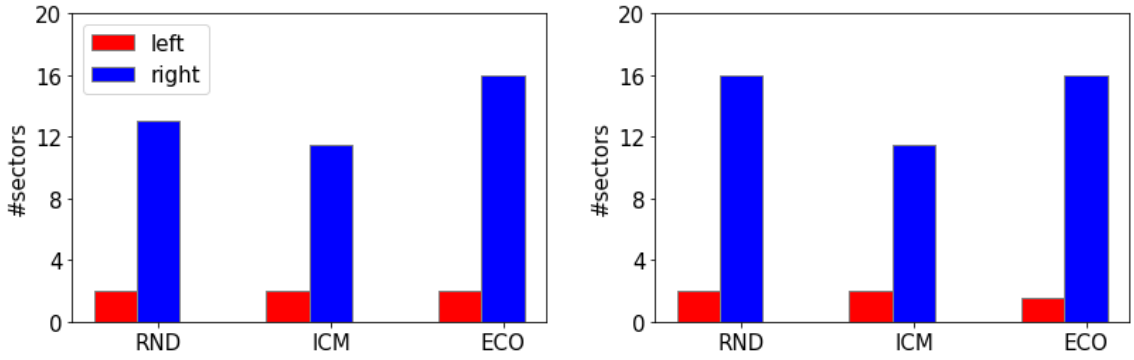
(a) Harmless scenario.



(b) Deadly scenario.

Figure 6.2. Visited sectors per episode. These four curves report the average number of sectors visited within an episode by the interacting agent after grabbing the key. Episodes when the key was not collected do not add information. For each of our two environments, these tallies are further group into sectors located to the left (17 in total) and to the right of the key (16 in total). Reported averages are computed by first taking cumulative averages within each run and then calculating the mean and variances of the former results across all 3 runs.

to evaluate a median distance between the current observation and those stored in the memory buffer. This means that every positive reward will have a value of $+1$, regardless of it is the first or n-th time we have seen a certain observation; at the level of a single reward there is simply no distinction. On the other hand, because of the nature of the percentile, the interacting agent can receive multiple consecutive bonuses even while not transitioning to a new state between time steps. And the exact amount of incentives that can be collected in a row at a given instant depends on the current size of the episodic memory and on how many times a specific observation has been registered. For example, when such a memory has size 2, an agent perceiving an observation whose reachability is 0.0 with respect to every entry in said buffer would get only one consecutive bonus. But in this very situation and with a memory of size 15, it would get three bonuses in a row. Furthermore, let's consider one last scenario where the episodic memory once again stores 15 elements



(a) Harmless scenario.

(b) Deadly scenario.

Figure 6.3. Average number of sectors discovered per run after the key was collected. These counts are additionally divided according to whether a sector is located to the left or to the right of the key.

and the agent teleports back to a state whose observation was added to this buffer back when it had 2 elements and which has not been re-visited ever since. On this occasion, our agent will be handed only two consecutive incentives. Note that this agent is not required to stay in place to get these subsequent bonuses, and in fact, it will necessarily get them anyways when heading forward. Taking all of this into consideration, anytime an agent decides to double back after been moving straight ahead, it will see a gradient in the number of consecutive bonuses that can be collected each time. This figure will be very low or even zero in areas near its present location, but it will grow as the agent gets closer to its initial position. Conversely, if instead our agent keeps journeying forward, such a number will start already as high as possible and will not ever decrease. Therefore, it is strictly at this level, in terms of cumulative rewards, that we see the bias of ECO toward uncharted parts of the environment that was mentioned in Subsection 6.5.1. Knowing this, we run a few additional experiments with the frozen reachability networks resulting from various training sessions. In these, we manually controlled the agent and steered it straight towards the key. Then, we sometimes kept moving right and others we went back to the left, stopping in both circumstances after being awarded a singular bonus. Thanks to these trials, we find out that when ECO attempts to retrace its steps it secures its first bonus after performing around 30 optimal actions. In contrast, when it relentlessly marches into unexplored areas of the world, it takes it less than 5 actions before collecting its first incentive, and within the span of 30 optimal actions it is awarded a total of 5 bonuses on average. In the end, we esteem that it is this marked discrepancy in reinforcement on the grounds of the direction of motion that prevents ECO from re-exploring. To this method, it is just significantly more rewarding to continue venturing into the unknown.

6.7 Discussion

Quite clearly neither RND, ICM or ECO are capable of doing re-exploration. So, what could we use or do differently in order to obtain a principled solution to our proposed environments? A key insight is that all three benchmarked methods could easily and rather efficiently solve these very tasks, especially the harmless scenario, if only they would be endowed with full observability of their world. Under such a condition, after discovering the key during a training session, they would proceed with their exploration in both directions, right and left, with equal interest and frequency. And since we know they can learn to navigate all the way to the rightmost room, such a uniform scouting should be guaranteed to eventually uncover the jewel present in our domains. Looking into the future, we would like to see adaptations of the former algorithms that would operate in much the same manner; as if partial observability was non-existent. Taking nature as reference, we do not recognize any compelling reason as to why this would be not possible, seeing that humans and animals have a even more advance usage of memory, which equally plays a critical role in exploration tasks. As a fairly abstract scheme, we envision the previous methods appending to their current observation another piece of data, which comprises their recollection of past events that occurred during the unfolding episode. Afterwards, this combined information is sent to their existing exploration modules for a business-as-usual computation of novelty or curiosity that however internalizes a partially-imagined complete state of the world. Notwithstanding, this is easier said than done. For example, recurrent policies, as the ones employed in this work, already maintain a state vector that in some fashion summarizes all events experienced following the last reset of the environment. Yet, using this vector directly as the memory component of the previous scheme is unlikely to succeed for two main reasons. One, this vector is not static, its meaning is constantly evolving and may not necessarily stop doing so. This creates confusing signals; as a point within this vector space may represent one observation before an update and a completely different one afterwards, but its associated bonus would remain linked to the prior observation for a while. And two, we have no guarantees that this vector will represent all the meaningful information of the world. For instance, it is implied by the results of the previous section that the key was irrelevant to the behavior of the three techniques that were evaluated. Thus, they would lose nothing by excluding said key from their internal states. All in all, this research route seems to be taking us into the realm of automatic feature extraction, which is another open problem within DRL. Therefore, we might wish to solve that one first, before applying its developments in our environments. A more plausible idea, particularly in small domains like the one used in this benchmarking, is to express our predictions of the entire state of the world directly as images instead of relying on a reduced vector representation. For example, a single prediction would be an image such as the one depicted in Figure 6.1(a), or contain only the rooms discovered thus far during the training session. Then, these full pictures are used in the calculation of bonuses in replacement of current observations. This automatically solves the previous two problems, as we will be trying to predict everything with no loss of information. Plus, we will now have the chance of grounding this representation on real-world observations, and that should instill a clear convergence course. Nonetheless, even in reference to

this proposal there still exist many uncertainties that require a deeper research. To voice a few: How do we construct examples if we never see the entire environment at once? Should we introduce assumptions that may not apply every time, such as the notion of object permanence: the world does not change unless the agent changes it? How do we even automatically construct examples? How do we define the size of the image representation if once again we are unaware of the full extent of the world? Should we let it grow on its own over time and bring in new neural capacity in response to new discoveries being made? How should an ever-expanding representation be handled by the bonus computation? How the acquired knowledge of novelty, curiosity or reachability for a 2-room world be transferred to a 3-room world?

Chapter 7

Benchmarking Reusability-Sensitive Sample Efficiency

Numerous deep reinforcement learning (DRL) algorithms have been proposed in recent years to handle sparse-reward environments. However; these, as it is common with most DRL approaches, demand a large number of samples to achieve good performance. This issue is more notorious in scenarios where skills must be reused several times within the same task. There, a learner able to quickly generalize a newly acquired skill could exploit this to accelerate exploration and to reduce its overall need for data. In this context, this chapter performs a much needed systematic analysis of the sample efficiency that is experienced in such scenarios by methods designed to deal with sparse rewards. Concretely, we benchmark three influential algorithms in a specially engineered sparse-reward environment that highlights the benefits of reusing intricate skills. We further devise a domain-specific indicator that puts a number on how much these methods leverage reusability to reduce their data hungriness. We finally report that, based on this indicator, state-of-the-art algorithms do not exploit skill generalization to any noticeable extent, which makes them severely sample-inefficient.

7.1 A Long-Lasting Desire to Learn Faster

Besides reward sparsity, which is the main topic of this thesis, sample efficiency is yet another major unresolved issue in DRL. Several recent studies show that state-of-the-art algorithms still demand an excessive amount of samples, i.e. training interactions with a given environment, to reach a satisfactory performance. Returning to the previous success cases, RND, NGU and Agent57, they all need over 20 billion (2e10) frames of experience before completing the first level of Montezuma's revenge. To put this into perspective, considering that Atari 2600 games run at 60 fps in their original console, such amount is equivalent to roughly 10 nonstop human years. Even though it is an unfair comparison since video games are specifically designed to evoke everyday human experience; a child would achieve the same performance in about two hours. OpenAI Five [71] is another example worth commenting. Impressively, this agent achieves superhuman performance in the game Dota. Nevertheless, it collected 180 human years worth of experiences per day during 10 months, approximately

50 thousand years in total, to reach this level of proficiency. These humbling facts aside, numerous works have lately tackled sample efficiency from different angles [108, 25, 19, 54, 53, 92, 121, 114, 62]. And the resounding progress shown by such efforts tell us that we are steadily inching closer to a definite solution.

In this work, we investigate one of the traits we posit a highly efficient learner must have. Namely, the ability to quickly reuse skills, i.e. goal-oriented behaviors extending multiple interactions steps. Here, *quickly* is synonymous with from few examples (previous realizations of a skill). In particular, we delve into the question of how sample-efficient state-of-the-art algorithms are while dealing with sparse-reward environments favorable to skill generalization. And we center this analysis on exploration-demanding skills, i.e. skills extending a great many unrewarded time steps. Far too many to be found by trivial exploration policies, which emphasizes the necessity of specialized algorithms that are robust to sparse rewards. Such domains are commonplace; a notorious case being Montezuma's revenge. There, an agent must recurrently enact the skill of picking up a key and unlocking a door with it. Critically, it receives a reward only when it grabs the key or opens a door, but nothing in intermediate states. Thus, on its own, this skill already poses a hard sparse-reward problem.

7.2 Contributions of this Benchmarking

As far as we know, there are no existing benchmarks that perfectly suit our needs in terms of functionality and simplicity of analysis. MR itself has several other challenges orthogonal to our research interest. Therefore, our first contribution is the proposal of a novel sparse-reward environment, crafted as a stripped-down version of MR. Its calculated design additionally allows us to easily isolate and measure what we called reusability-sensitive sample efficiency (RSSE). An indicator that gives a concrete answer to our original question. As a second contribution, we perform a benchmarking in which we test three prominent algorithms in this environment and estimate their associated RSSE. Namely, we experiment with: RND [12], ICM [74] and ECO [84], which are chosen specifically because of their known competence in dealing with sparse rewards. The RSSE scores resulting from these experiments give no indication that these algorithms take advantage of skill reuse at any level during exploration. In fact, they inefficiently need many times more samples to replicate a skill than to discover it. An outcome that reveals that there is still a lot to be done in this area. One final contribution we offer is the pointing out of three unconnected research directions that have a good opportunity of making progress toward a higher sample-efficiency. Namely: active exploration, hierarchical learning and advanced reasoning.

7.3 Related Work

7.3.1 Approaches to Sample Efficiency

Sample efficiency has seen substantial improvements with varying degrees of success originating from diverse research directions, such as: careful tuning of training

hyperparameters [108], improving generalization in neural networks [25, 56], learning compact and meaningful representations that facilitate policy training [19], optimizing additional losses from domain-independent auxiliary tasks [54, 89], automatically augmenting the dataset of observations [53, 78], pursuing directed exploration [92, 90], embracing model-based reinforcement learning [121, 118], learning hierarchies of agents or skills [114, 117] and integrating advanced reasoning into DRL [62, 119].

7.3.2 Similar DRL Benchmarks

Next, we report a number of benchmarks that, although not being a perfect fit, have some similarity to ours.

Inspired by Montezuma's revenge: Toy MR [83] and Montezuminha [10] are also gridworld domains influenced by this Atari game. Toy MR reconstructs with great detail the entire first level of the game; thus, like the original version, it is unnecessarily complex for our needs. Montezuminha would have been a good choice, except for one concern: a single color is used to denote both keys and likewise both doors. Hence, it might not be too clear to a learner that the remaining key may open the remaining door, as it likely already failed to unlock such door with an identical key.

Favorable to skill reuse: There are plenty of benchmarks in the literature where it is possible and beneficial to generalize skills. A few that have been regularly employed include: four rooms [104], food collection [20] and block stacking [52, 17]. Unfortunately, none of them aligns with our purposes. In its standard implementation, the former one does not comprise exploration-demanding skills. On the other hand, the latter two allow different executions of the same skill to be learned in parallel. Such concurrency makes it difficult to analyze how much faster or slower subsequent skills are acquired relative to previous ones.

Focused on efficiency: Procgen [16] and Alchemy [113] are two frameworks that target sample efficiency. They rely on procedural generation to evaluate how well a learner trained in a large sample of tasks generalizes to new tasks instantiated from the same distribution. However, this is contrary to our constraint of generalizing from few examples. Otherwise, the right instantiation of the domain *Heist* would have been suitable for this work.

7.4 Environment Design

To fulfill our research interest, we have specifically designed a sparse-reward environment that highlights the benefits of skill reuse. Our environment takes inspiration from Montezuma's revenge; namely, we borrow its reward scheme, its gameplay of finding keys to open doors, as well as its room-based layout. And we integrate all these elements into a cut-to-the-bone version of the game that shaves off many other difficulties to focus solely on the interplay between sparsity, reusability, exploration and sample efficiency. We name this environment: two doors & two keys (D2K2).

In Fig. 7.1, screenshoots from 1 to 9 depict one possible trajectory that solves our domain, which we describe next. Our task has an agent starting in the left-side room of a two-room world. There, it must pick up a blue key and with it unlock a blue door, while avoiding deadly pits (red cells). Only after doing that it can access

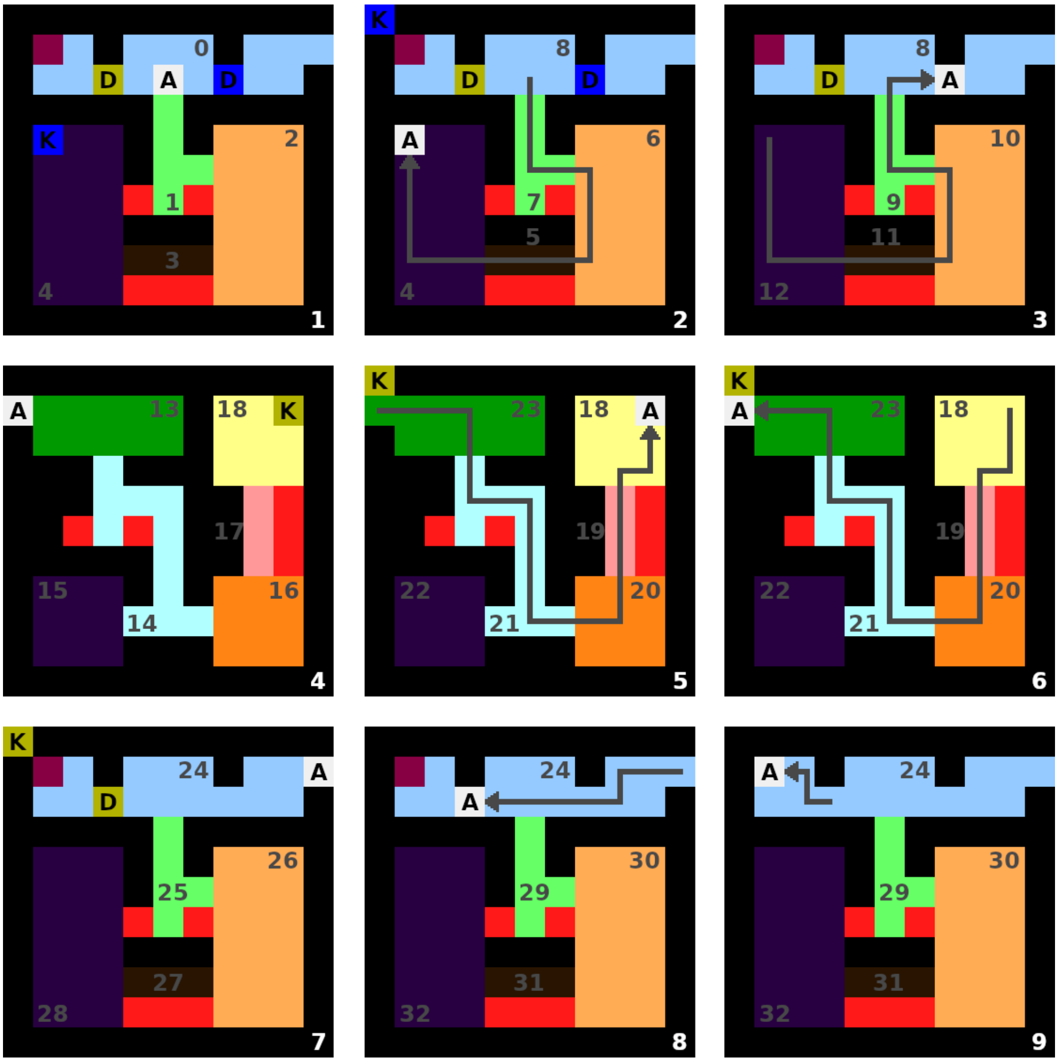


Figure 7.1. Screenshots of D2K2. *A*, *K* and *D* stand for agent, key and door, respectively. Sectors (same-color regions) are labeled with numbers inside or next to them. And they are renumbered after important events.

the right-side room, where it must find a yellow key. Finally, it needs to return to the left-side room to collect a jewel (burgundy) that lies beyond a yellow door. This jewel grants the only positive reward (+1) of the environment and the episode terminates when the agent grabs it. The usefulness of reusing skills is evident here. To solve the task, the agent must concatenate twice the general skill of unlocking a door after picking up the key of the same color. This generalization is sensible because by construction the agent experiences basically the same sort of difficulties both times it performs this maneuver (e.g. navigation dynamics, arrangement of deadly pits). Therefore, after opening the blue door, an algorithm embracing reusability is expected to replicate this skill without delay to open the yellow door, as it continues to explore the environment. And by doing so, it is very likely to find the jewel relatively quickly.

Next we provide extra details about the proposed environment. Its state consists

of two $11x11$ grids that represent two contiguous rooms. Within these rooms, we find three types of static elements: walls, pits and spots. An agent cannot go through walls, dies if it falls into a pit (episode terminates) and is free to visit any spot. Spots are nominally grouped into several sectors, but there are no functional differences between them. On top of that, this domain accommodates a variety of dynamic elements, including: an agent, two keys, two doors (initially locked) and one jewel. During an episode, these elements can move between cells or even disappear entirely from the state. Specifically, the agent is allowed to move to any adjacent spot or pit (but it cannot pass a locked door). It collects items (keys, jewel) and unlocks doors just by moving to their cells (to open a door it must also hold the proper key). As soon as a key is grabbed by the agent, it is relocated to the upper left cell of the room in which the agent is presently found (they travel across rooms together with the agent). Meanwhile, collected keys, doors and the jewel are removed from the state when used, unlocked or grabbed by the agent, respectively.

The observation space is a composite of two images corresponding to two independent renderings of the internal state. Importantly, both offer a full view of only the room the agent is currently in. As seen in Fig. 7.2, one image can be thought of as a standard view; there, colors are assigned somewhat arbitrarily to each element in the environment. Crucially, in this view, every door is depicted with the same color as the key that unlocks it. The other image describes a semantic view, in which colors are uniquely assigned to classes of elements (e.g. spots, keys, doors). The exact specification of the observation space ultimately depends on the researcher, since our practical implementation of the environment provides multiple options. The color format of the standard view can be configured to be either RGB or grayscale (the semantic view is always rendered in grayscale). The smallest resolution is obviously $11x11$, but there is no limit or constraint on the maximum height or width. Likewise, there is no limit on frame stacking. Nevertheless, in this work, observations have comprised, depending on the algorithm, either a single frame or two consecutive frames of $44x44x2$ grayscale images.

Moreover, the action space of D2K2 is discrete with a set of 4 actions denoting the four cardinal directions in which the agent is allowed to move. There are no special actions for grabbing or unlocking. To make this domain more challenging, we add a source of stochasticity implemented in the form of sticky actions. As usual, this implies that with probability p the latest action selected by the agent will be ignored and the action taken in the previous step will be executed instead. In all our experiments, we have fixed this probability to 0.25.

A few final remarks. The initial location of every dynamic element remains constant across episodes (including the agent). As already mentioned, only the jewel grants a positive reward and there are no other incentives of any kind in D2K2 (positive or negative). This is a minor discrepancy with respect to Montezuma's revenge, where gathering keys is also substantially rewarded. An episode terminates early when the agent discovers the jewel or when it falls into anyone of the ten deadly pits. Otherwise, each episode has a maximum duration of 800 interaction steps.

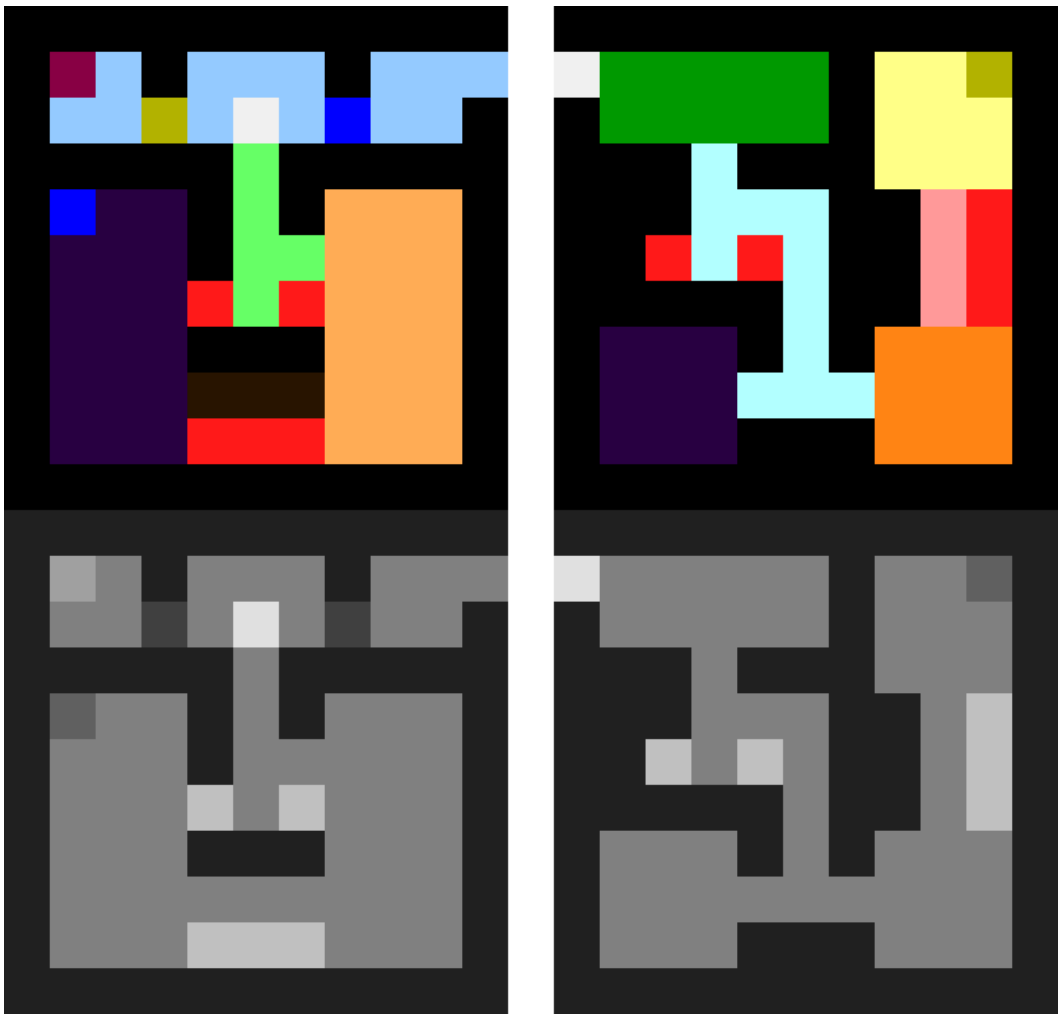


Figure 7.2. Observations rendered when an agent starts an episode (left) and when it first enter the right-side room (right). Each comprises two images: standard view (top) and semantic view (bottom). Color code (standard view): walls (black), pits (red), agent (white), keys (blue/yellow), doors (blue/yellow), jewel (burgundy), spots and sectors (any other color)

7.5 Benchmarking Setup

This section elaborates on a benchmarking effort, in which three influential DRL algorithms proven to handle sparse-rewards are confronted with our proposed environment.

7.5.1 Covered Algorithms

The three approaches taking part of this benchmarking are those contained in Subsection 3.3, specifically: RND, ICM and ECO. A quick remark concerning ECO, its intrinsic reward scheme is modified slightly here. Specifically, observations entering the episodic memory grant a bonus of +1, while every other hands a bonus of zero.

Beforehand, we do not expect any of these three algorithms to produce a top-notch sample efficiency. Within their formulations, we simply do not see any proper mechanism that would allow them to identify, store and reuse exploration-demanding skills. The singular faculty they have for generalization is the one intrinsic to their neural models, but this is unlikely to be of much help at the level of skills. In fact, it would come as a great surprise to us, if any of these techniques takes relatively less time at learning to open the yellow door than the blue one.

7.5.2 Training Details

A similar policy-value network is employed by all benchmarked algorithms. Such policy consists first of a stack of 4 convolutional layers with 32 filters each, kernel size of 3, stride of 2, same padding and RELU activation. Afterwards, within the implementations of RND and ICM, the information coming from this convolutional module is flattened and passed through 4 dense hidden layers with RELU activation and 512 units each before going to the policy and value output heads. On the other hand, in the case of ECO, information flows from the flattened convolutional output to a single LSTM layer composed of 512 units and finally to the output heads. An explanation of why we use a recurrent architecture exclusively with ECO is provided in Section 7.6. Additionally, RND uses convolutional modules of the same topology for its random and predictor networks, but instead their activations are leaky RELU. Each of their outputs then goes through 2 dense hidden layers (512 units/layer and RELU activation) and a linear output layer with a representation size of 256. In like manner, ICM adopts this exact 4-layer convolutional assembly for the encoder of its curiosity module, except that it replaces the activations with ELU. Furthermore, its forward and inverse models are each built with a single dense hidden layer containing 256 RELU units; plus, the former model has a linear output layer whose dimensionality is 288. For its reachability classifier, ECO builds a siamese network where each of its two branches has a replica of the convolutional module being used by its policy. Notably, both networks receive only the latest 2-image frame of the observation as input. These branches are followed by a stack of 4 dense hidden layers with 288 RELU units each and, subsequently, by a softmax layer with 2 outputs.

Table 7.1 gathers most of the remaining hyperparameters used across this benchmarking.

7.5.3 Sample Efficiency Evaluation

In this benchmarking, we want to quantify the sample efficiency associated with skill reuse. The way we approach this is by estimating how many extra samples a learner needs before generalizing a previously discovered skill. From section 7.4, we know that, to solve our D2K2 task, an agent must execute twice the same general skill, i.e. it must unlock first the blue and then the yellow doors. Then, to evaluate such efficiency, we simply count the number of samples taken during a run between when a learner first realizes it can reuse the unlocking skill and when it first reuses it. Specifically, these two milestones correspond to the first time a learner enters the right-side room and to the first time it unlocks the yellow door, respectively. Note

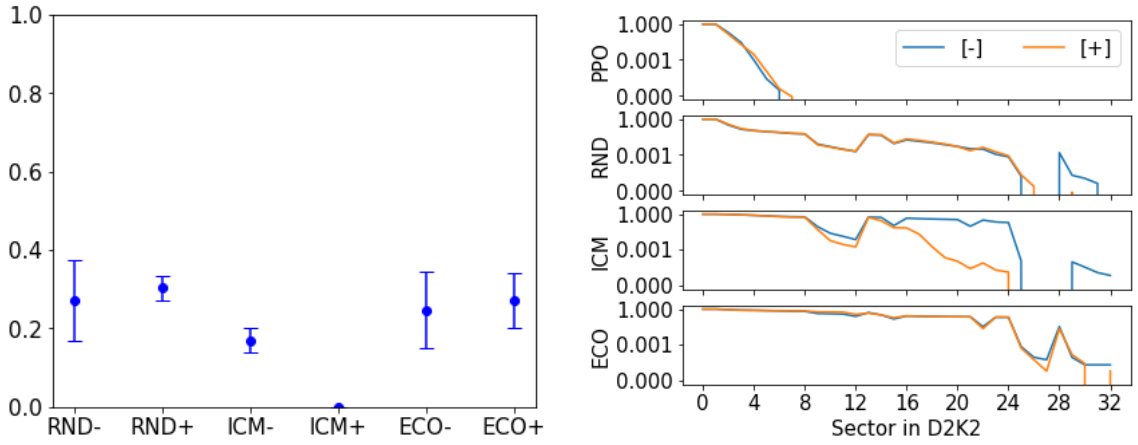
Table 7.1. Overview of applied hyperparameters

Hyperparameter	PPO	RND	ICM	ECO
Number of workers	16	32	16	16
Frame stacking	1	1	2	2
Rollout length	128	128	80	256
Number of optimization epochs	4	4	4	4
Learning rate	1.0e−4	1.0e−4	1.0e−4	2.5e−4
Entropy coefficient	0.01	0.01	0.01	0.1
PPO (λ)	1.0	1.0	1.0	0.95
Coefficient of extrinsic rewards	-	2.0	1.0	1.0
Coefficient of intrinsic rewards	-	1.0	0.01	0.03
Extrinsic discount factor (γ_E)	0.99	0.999	0.99	0.99
Intrinsic discount factor (γ_I)	-	0.99	-	-
RND experience fraction used by predictor	-	1.0	-	-
ICM curiosity loss strength (λ)	-	-	10	-
ICM forward inverse ratio (β)	-	-	0.005	-
ECO memory size	-	-	-	320
ECO reward shift (β)	-	-	-	0.0
ECO novelty threshold ($b_{novelty}$)	-	-	-	0.3
ECO aggregation function (F)	-	-	-	P_{90}
ECO reachability training learning rate	-	-	-	1.0e−3
ECO reachability training history size	-	-	-	128K

that an algorithm cannot enforce reusability in a sensible manner until not reaching the right-side room and seeing the yellow key. Thus, before that point, we cannot meaningfully quantify its related efficiency. By now, such count of extra samples is already a working measure of efficiency from a reusability perspective. According to it, a maximally efficient learner requires zero extra samples. Conversely, a learner that gathers an infinite number of them is minimally efficient. Nevertheless, we further normalize this measure to make it more informative. In particular, we divide it by the number of samples taken by an algorithm until it first opens the blue door. In this way, for our specific domain, any value lower than one surely indicates that a learner is exploiting skill reuse to some degree. Moreover, we rescale and invert the range $[0, \infty]$ to turn it into $[0, 1]$ (higher value→higher efficiency), which is more proper for an efficiency measurement. In this transformation, we force 1.0 to become 0.5 (for better readability), so that now any result above the latter value is predictive of a greatly efficient algorithm. Finally, after taking all the previous considerations into account, we formulate the following indicator:

$$RSSE = 1 - tanh(0.55 \cdot (\frac{S^{yd} - S^{rr}}{S^{bd}})^{0.35}) \tag{7.1}$$

Where, RSSE stands for reusability-sensitive sample efficiency. S^{bd}, S^{yd} and S^{rr} correspond to the number of samples taken before unlocking the blue door, unlocking



(a) RSSE scores.

(b) Historical sector visitation ratios.

Figure 7.3. Benchmarking results. Symbols [−] and [+] denote settings without and with sticky actions, respectively. Graph (a) reports mean RSSE scores across runs and their associated standard deviations (PPO is excluded because its RSSE is undefined in both settings). Graph (b) reports, in logarithmic scale, mean ratios across runs of the number of episodes each sector of the D2K2 domain was visited by the learning agent over the total number of episodes completed during a run.

the yellow door and reaching the right-side room, respectively.

Note that if a learner never arrives at the right-side room, then we will have $S^{yd} - S^{rr} = \infty - \infty$, making its RSSE undefined. In compliance with what was said earlier, an undefined result simply means that we do not have enough information to estimate the RSSE of a learner that has never seen the yellow key. The same situation occurs if a learner is unable to unlock the blue door. Furthermore, if throughout a run a learner reaches the right-side room at least once but it always fails to unlock the yellow door, we will end up with $(S^{yd} - S^{rr})/S^{bd} = \infty$. In this case, we consider that the corresponding RSSE approaches zero in the limit.

7.6 Results

We benchmarked RND, ICM and ECO on our D2K2 environment. PPO was evaluated as well, but only to verify if rewards are sufficiently sparse in our domain. We further considered two settings: with and without sticky actions, to get a better understanding of the effects of stochasticity in each algorithm. All eight experiments were repeated with five different seeds to assess variability and each run lasted for 20 million interaction steps (frames).

From Fig. 7.3(b), it is clear that rewards in the D2K2 environment are too sparse for PPO to handle. So much so that this method never opened the blue door, not even once in any of the ten runs it partook. An outcome that we take as confirmation that our self-imposed prerequisite of designing an environment in which learning a single skill already poses a challenging sparse-reward problem is satisfied. With regard to the three benchmarked algorithms, as reported in Fig. 7.3(a) and Table 7.2, none of them achieved an RSSE above 0.5 in a single run, which

would have been indicative of a timely reuse of skills. In fact, they all procured an RSSE on average lower than 0.33; in other words, they required at the very least three times as many samples to repeat a skill than to first discover it. Moreover, just the experiment involving ICM and the presence of sticky actions produced a mean RSSE of zero, which refers to cases in which the given learner never managed to unlock the yellow door. Our results demonstrate that this learner is negatively affected by this kind of stochasticity; unlike the other two methods, which seem indifferent to it. Such outcome is not at all surprising considering how this algorithm computes intrinsic rewards. Since the existence of sticky actions makes it impossible to perfectly predict both forward and inverse dynamics, ICM remains permanently curious about anywhere in the environment during a run. Note also that this method cannot just ignore this source of unpredictability as it would do with others that are not under the control of the agent (e.g. leaves moving with the breeze). In the end, such ubiquitous never-ending curiosity makes this learner reluctant to explore the D2K2 environment, as it would still be non-negligibly rewarded for staying in place and avoiding an early death. Therefore, what we finally observe is that ICM progresses much more slowly through this domain than through its deterministic version and it is potentially unable to effectively solve it. We consider valid now to explain why ECO did not used a feedforward network as policy. The short answer is that when having such a policy, this method becomes trapped in the initial room and never solves the task. To be more precise, this algorithm has the misfortune that opening the blue door creates a bifurcation in the D2K2 domain. At this junction, an agent can either head towards the yellow key or towards the location where the blue key was previously found. Ultimately, ECO always explores the latter path faster; presumably because it is almost identical to the one it has just taken. Indeed, it finds sector 12 much sooner than sector 14 (see Fig. 7.1); and this fact has a profound effect on the final behavior of this algorithm. Once it reaches sector 12, it has minimal interest to take the other path at the bifurcation, since moving to such a sector already provides a considerable reward. And this would be forgone if trying something else without comparable success. Furthermore, in contrast to RND and ICM, the bonuses of ECO do not wither down with visitations; therefore, there is little chance that it will get bored with this left path. Interestingly, because ECO has some incipient backtracking capabilities, there should presently be sizeable bonuses along the path connecting sector 12 back to the former position of the blue door. And this is the critical point; a feedforward policy, unlike a recurrent one, cannot collect such bonuses. It simply cannot retrace it steps, as that would require a memory that its neural architecture does not have. In consequence, an unretentive ECO gets forever stuck in sector 12, which acts as a local maximum within its reachable bonus landscape.

7.7 Discussion

Qualitatively speaking, that these three algorithms presented a somewhat low RSSE was not completely unexpected, since they were not designed with sample efficiency in mind. In fact, none of these methods possesses any sophisticated mechanism for boosting generalization other than the inherent abilities of neural networks. However,

Table 7.2. Data contributing to the computation of RSSE scores.

Algorithm*	#samples as defined in Eq. 7.1 (in millions)			RSSE
	S^{bd}	S^{rr}	S^{yd}	
RND−	2.7M±1.1M	3.8M±0.5M	15.6M±2.9M	0.27±0.10
RND+	3.1M±0.9M	4.6M±1.2M	15.5M±2.3M	0.30±0.03
ICM−	1.4M±0.4M	1.7M±0.3M	14.3M±4.8M	0.17±0.03
ICM+	1.3M±0.4M	1.5M±0.4M	∞	0.0
ECO−	1.4M±0.5M	1.5M±0.5M	9.5M±2.4M	0.25±0.10
ECO+	1.8M±0.5M	1.9M±0.6M	10.8M±4.7M	0.27±0.07

*Symbols [−] and [+] denote settings without and with sticky actions, respectively.

these are simply not enough for scenarios such as the D2K2 domain, where, given a single example, an efficient learner is supposed to immediately reuse a skill in an unfamiliar part of the world. Another common flaw that we have identified in all these learners is that they strictly follow a fine-grained incremental exploration strategy. In other words, they try to visit every nook and cranny of the observation space while never straying too far from already visited regions. And this is the result of two choices made by these algorithms: to operate only at the level of primitive actions and to depend only on randomness to discover new regions of the environment. Obviously, such meticulous and cautious approach to exploration is perfectly sensible and broadly competent. But it can be rather slow in many circumstances, especially when there exist generalizations that are not being exploited. Furthermore, these issues are not exclusive to bonus-based exploration methods, as the ones examined in this benchmarking. Instead, they permeate through nearly all research avenues and algorithms exposed in Subsection 3.2. Indeed, of the best-performing algorithms in domains with very sparse rewards (e.g. Montezuma's revenge), none has emphasized sample efficiency and they all adhere to the paradigm of a gradual exploration.

Two notable exceptions are Plan2Explore [90] and FuN [111]. Plan2Explore replaces the passive discovery of novel transitions and states with active exploration. That is, this learner consciously decides which not-yet-visited regions of the world to travel to next and plans trajectories to reach them. Even though Plan2Explore still searches through an environment gradually, the idea of active exploration is not limited to that. As a matter of fact, it would be highly desirable to have algorithms able to fulfil long-horizon exploration goals. In particular, in the D2K2 environment, we would like to see learners that after getting one glimpse of the right-side room decide to explore beyond the yellow door, while ignoring every other sector in between. In turn, FuN embraces the concept of hierarchical reinforcement learning. Hierarchization would likewise be useful in our proposed task because it extends an agent's action set beyond primitives to include learned skills. An this has the glaring potential of accelerating the pace of exploration. Nonetheless, more work needs to be done to implement this concept satisfactorily. For example, FuN compares poorly to state-of-the-art methods in Montezuma's revenge in terms of both performance and efficiency. Besides active exploration and hierarchical learning,

a third idea we believed essential for solving the D2K2 problem efficiently is the integration of advanced reasoning into DRL. An algorithm enriched with tools such as causal, inductive and analogical reasoning would envision helpful generalizations on the fly that, for instance, could serve as guidance for an active explorer.

Chapter 8

Benchmarking Exploration Bonuses in a High-Precision Robotics Task

Deep reinforcement learning (DRL), aided by bonus-based exploration methods, has made significant advances in recent years toward dealing with sparse-reward environments. However, successful algorithms have been tested mainly on domains whose state space can be compressed without lack of performance into a relatively small number of regions. Thus, one lingering question is how such techniques will respond to more complex environments featuring a ridiculous amount of distinguishable regions even after a reasonable state abstraction. To answer this concern to some degree, the present chapter puts forth a new sparse-reward benchmark. Namely, a domain framed as a synchronous dual-arm manipulation task that, due to the high-precision required for solving it, possesses the aforementioned property. In this novel environment, we evaluate a few state-of-the-art algorithms proven to handle reward sparsity by virtue of intrinsic bonuses. Ultimately, this study reveals that the examined methods emphatically fail to address the proposed domain; as matter of fact, they never once completed our task. Their downfall being triggered by three factors: their formally unbiased approach to exploration, the vastness of the presented benchmark and the limits of modern computing hardware. We then conclude this work by discussing the prospect of nevertheless relying on reward bonuses to tackle scenarios comparable to ours and what changes could be made to these incentives to that end.

8.1 Size of the Abstract State Space Matters

One general framework for dealing with reward sparsity embraces the notion of bonus-aided exploration. A learner following such approach internally computes extra rewards, named bonuses or intrinsic rewards, which are non-zero in numerous states and which are handed to the interacting agent on top of the task incentives (also known as extrinsic rewards). These bonuses are necessarily derived from observations of the world, which are the only source of information left in the absence of rewards. Plus, for the sake of being broadly applicable, they are always

designed in a very general fashion with no explicit preferences for any relevant part of an environment. Under an ideal operation, such intrinsic incentives help circumvent reward sparsity by promoting the visiting of every state, and ergo, the detection of every (state-bound) task reward. This strategy has been internalized by several algorithms and their remarkable results prove its utility. For example, RND [12] reports traversing all 24 rooms in the first level of Montezuma's revenge (MR), a paradigmatic sparse-reward problem in which general-purpose methods are incapable of escaping even its first chamber. Moreover, ICM [74] and EC [84] proclaim solving visual navigation tasks that are unmanageable for a basic DRL method such as PPO [88], which does not put any special attention to reward sparsity. Not only that but these two are additionally some of the very few techniques able to conquer domains riddled with distractors the likes of a broken or a noisy TV. Yet, an important detail about the former algorithms and their success stories concerns the environments where they were tested. These exclusively had state spaces that could be compressed into a modest number of abstract states over which exploration bonuses still worked well enough to not miss any extrinsic reward. For example, we have estimated that MR can be partitioned into the not at all daunting figure of 1,600 macro states per game reward when considering only information relevant for navigation. Or the still manageable total of 10^7 abstract states per task reward when nothing visible on the game screen is left out (see details in Subsection 8.3).

8.2 Contributions of this Benchmarking

Given these antecedents, we were curious to see what would happen when bonus-based exploration was tasked to handle more gargantuan environments. Ones that even after a reasonable abstraction would possess an unfathomably large number of macro states per task incentive. To this end, as a first contribution, this work crafts a manipulation problem where a simulated bimanual robot is handed a singular reward only after simultaneously lifting a key and a padlock from a table and using the former to unlock the latter. Intriguingly, this task appears of low difficulty at first glance; certainly it is trivial for us humans, and many other animals can also learn it. Nevertheless, according to an approximate abstraction analysis done by us, this domain can be reasonably partitioned into no less than the exorbitant amount of 4.5×10^{35} macro states per extrinsic reward. An exploration space that, in line with our original requirement, is effectively several orders of magnitude vaster than any prior sparse-reward environment. Our second contribution is a benchmarking endeavor that examines the performance of a handful of accomplished exploration bonuses, namely RND, ICM and EC, in the previous robotics scenario. Results from this study reveal that even though it is palpable that bonuses improve the effectiveness of exploration, they are still utterly ineffectual at dealing with the proposed environment. The size of its abstract space is simply too massive that it would take many lifetimes for present-day machines to visit every one of its states. And this means an insurmountable road block for bonus-based exploration, which as mentioned earlier, to tackle reward sparsity, usually try to travel evenly everywhere within a domain until finding a task reward. As a result, in our experiments, we see that by force of probability all three algorithms spend most of their time searching

through state regions that do not lead to the unlocking of the padlock. In fact, after over 40k episodes per run and 3 runs each, none of them managed even once to have both objects lifted from the table at the same time. One final contribution is the discussion of where do we go from here and the outlining of stimulating research directions that in many cases consider the integration of human biases into the exploration behavior.

8.3 Related Work

Seeing that in Section 8.1 we expressed that algorithms relying of bonus-based exploration have only been tested in environments whose observation space can be sensibly encoded into a small number of abstract states, in the following we proceed to support this claim. We start this analysis by assuming that learners working with bonuses will usually be tuned to prioritize extrinsic over intrinsic reinforcement (which is the common practice). A direct consequence of this preference is that they will tend to develop policies that travel from one task reward to the next following the shortest path possible while ignoring any intrinsic incentive in between. Indeed, it will not be until past the last discovered extrinsic reward that they will pay attention to bonuses. In other words, from the perspective of said learners, a meaningful exploration will only happen independently within each observation subspace existing between two consecutive task incentives. And from such an outcome it further follows that, given our starting assumption, these subspaces are de facto more representative of the difficulty of exploration within an environment than the full observation space. In light of this, next we will be estimating the size of the abstract space associated with a given domain, i.e., the measure at the heart of our allegation, with respect to precisely said subpaces. Going through the corpus of works that investigated bonus-based exploration (see Subsection 3.2.1), we observe that the majority of them evaluated their intrinsic rewards in Atari games [97, 9, 106, 73, 98, 12, 49]. Besides that, other testbeds included the video game Super Mario Bros. [74], small 3D mazes [74, 84] and a handful of elementary control problems [106]. Among these, inarguably, the ones stemming from video games possess the most complex state spaces. Moreover, within them, Montezuma's revenge and Pitfall are objectively the ones requiring the most actions to move from one task reward to the next, which implies that the exploration spaces between any two of their incentives are likely the biggest of them all. Therefore, to validate our claim, we will be examining only Montezuma's revenge (MR), since it should define an upper bound on the quantity we want to estimate, plus it is the most famous problem out of this bunch. In this game, the most vital component of the internal state is the XY position of Panama Joe (avatar). This can be reduced to few macro states by dividing each room into a coarse grid and using the centers of the resulting cells as a proxy of the avatar's location. To define the size of the grid, we can replicate the partitioning mechanism employed by EC where abstract states are separated from each other by a fixed number of actions (usually 5). Since, within any chamber of MR, moving from far left to far right or from far up to far down and vice versa requires less than 25 actions (using a frame skip of 4), then applying the previous idea produces a $5x5$ grid. However, not every cell of every room is actually reachable by the

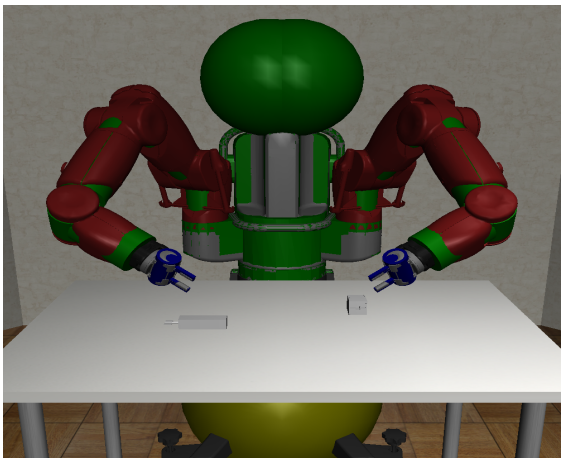
avatar; thus, instead of 25 macro states per chamber a more realistic average is 14. Another important piece of information is the XY position of moving creatures (skulls and spiders), and here we can repeat the above procedure with the same grid to clump multiple states together. There are 8 such creatures in the entire first level of MR, all displaying a much more restricted motion than the avatar, from which we estimate a mean of 2 macro locations per room. Furthermore, the initial level of MR also contains 33 elements (omitting those in the treasure room) that can take just two states: they are displayed on the screen or not. These include snakes, doors, disappearing floors, laser gates, tools and jewels; together they roughly account for 3 abstract states per chamber. To solve MR, one last thing a learner needs to know is which tools are currently in its possession, e.g. amulets, swords, torches and keys. This is shown in the informational section of the game screen (topmost); each tool there is represented by one icon, which gets duplicated when another widget of the same type is collected. Tools are removed from this inventory only when they are used or the timeout for holding them expires. Notably, these icons travel across rooms along with Panama Joe, meaning that a widget found in one chamber may increase the state count in all others. However, such tally does not grow too large because of the many pre-conditions MR has regarding captured tools, such as: no more than 4 being allowed in the inventory at any time or their icons being ordered in a row according to type. All in all, for the starting level of MR that contains 7 tools (4 of them being keys), we calculate that collected widgets can be grouped into around 20 different ways, thus, defining 20 additional macro states per room. Now, by multiplying the four previous factors and aggregating over 24 chambers we get merely about 40k abstract states per entire level. But this may not necessarily be the whole picture as the informational section of the game screen further harbors details about the number of lives remaining and the current score. Although such data is completely irrelevant for navigating the maze, since it takes diverse values during an episode, learners might regardless be motivated to explore these states. Remaining lives are depicted as hats; the game starts with 5 hats and one is removed from the screen every time Panama Joe meets its death. Since the avatar gets one more try after all hats are gone, this feature of the game raises the tally of abstract states by a factor of 6. Regarding the score, it keeps track of the sum of points earned by an agent during a full episode, which is displayed in big Arabic numerals. In MR, different amounts of points are awarded after completing various deeds, including: grabbing a jewel (+1000) or tool (+100, +200 or +3000), opening a door (+300) and slaying a creature (+2000 or +3000). By combining the previous rewards in all feasible ways we get a number of distinct scores in the ballpark of 1000 for just the first level of the game. At this point, putting everything together results in 2.5×10^8 abstract states per level. To finalize this analysis, we must lastly account for the density of task rewards visible in MR. Just in its first level we found around 25 of them whose location remains static across episodes, including: grabbing jewels or tools, opening doors and slaying monsters. From this, we can overestimate the size of the subspaces found between rewards by simply dividing the former two numbers, which produces an average of 10^7 macro states per subspace, or equivalently, per reward. This last figure may seem somewhat large, but in fact it aligns fairly well with previous results. Case in point, RND needs over 5 billion direct interactions with MR before learning to travel through 24 chambers of this game. A training

period so long that every one of our abstract states could be visited about 20 times before its end.

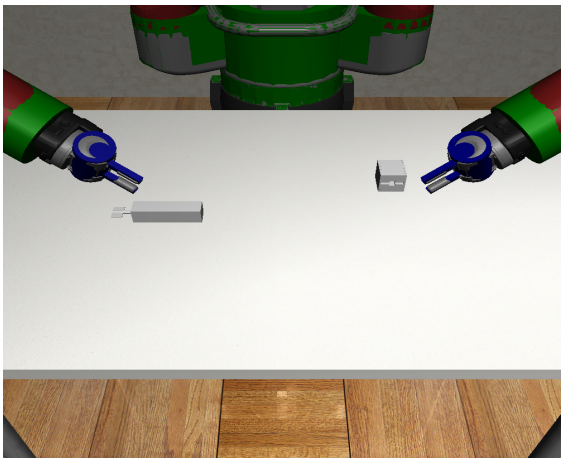
8.4 Environment Design

The new benchmark we propose in this work is a manipulation task in which a bimanual Baxter robot is rewarded only when it manages to open a padlock with a key (see Figure 8.1). Generally speaking, we expect this to be a rather motivating environment seeing that its reward structure is very sparse and its solution is quite intricate. Furthermore, in line with the main topic of the present study, this task is a great example of a domain whose state space can only be sensibly abstracted into a immense number of macro states, as we will see in more detail below.

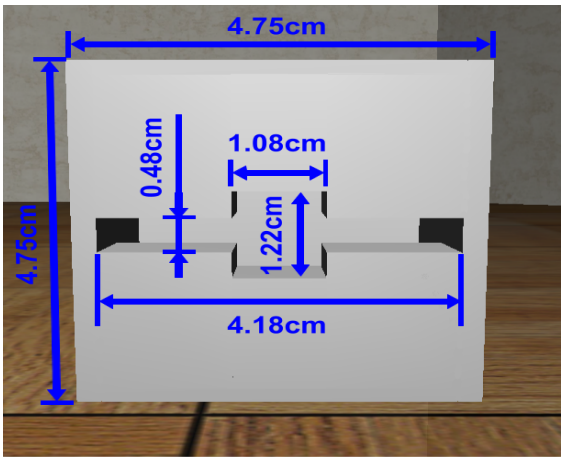
Going into more detail, this environment is simulated entirely in MuJoCo [107]. It contains four main elements: a robot, a table, a padlock and a key. The robot has two 7 dof arms and each possesses a 2-jaw parallel gripper as end-effector. For our domain, we restrict the motion of this robot such that only the joints belonging to its arms and grippers remain operational, which is also to say that the poses of its head, body and base are immutable. Likewise, the position of the table is fixed and cannot be changed during simulation. In particular, we place it right in front of the robot; close enough that every inch of its top surface can be reached by at least one gripper of the latter. As to the padlock and key, we fabricated them to resemble the real-life objects, except for the handle of the key, which is shaped as a square cuboid to facilitate grabbing. Needless to say, these are free-moving elements that can be relocated to any empty space according to similar physics as those acting over their worldly counterparts. In our proposed task, every episode starts with the robot, padlock and key at rest and always in the exact same poses and joint configurations (this initial state is shown in Figures 8.1(a) and 8.1(b)). The observations provided by this environment are in the form of a real-valued vector of length 60. This contains the absolute 3D position, orientation, linear and angular velocity of both end-effectors and both objects plus the between-jaws distance and linear velocity of both grippers. In terms of actuation, all 16 joints (14 revolute and 2 prismatic) are controlled through torque commands ranging from -1 to 1. Additionally, a frame skip of 25 is enforced on all occasions, meaning that each action is internally repeated that many times before returning control to the learning algorithm. It is worth mentioning as well that, due to the inherent noisiness of MuJoCo, this domain is not fully deterministic; hence, performing a specific action multiple times in a particular state will in general lead to different outcomes. Normally, these discrepancies will be minimal; except when collisions occurs, since their simulation is more chaotic. Our environment awards a positive incentive (+1) solely in one situation: when the robot completes the task of unlocking the padlock, and in all other circumstances it grants a reward of zero. Crucially, to achieve such a goal, the robot must priorly follow a short series of steps. First, it must lift the padlock and key from the table; as otherwise, because of the dimensions of these objects, it would be virtually impossible to execute the next steps, specially the last one. Second, while concurrently holding both objects, the robot must insert the key into the keyhole. And third, after a successful insertion, it must rotate either



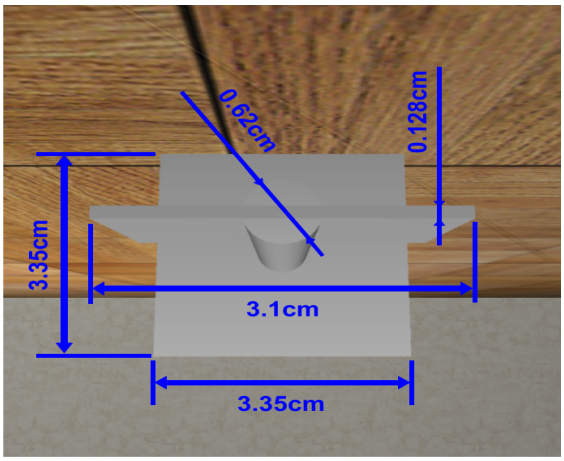
(a) Front view.



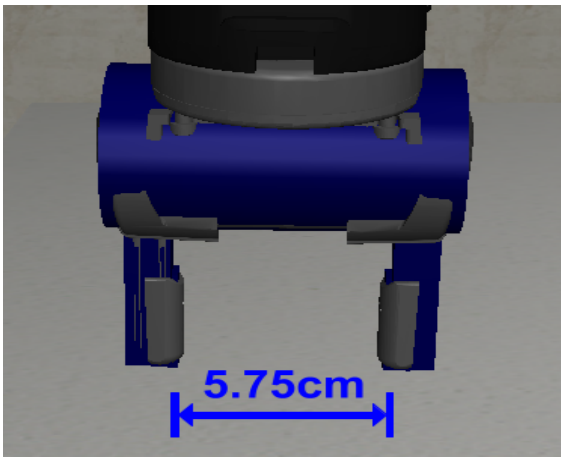
(b) Agent view.



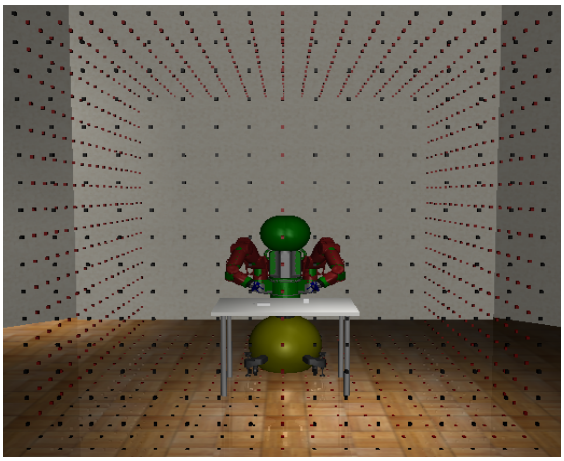
(c) Padlock.



(d) Key.



(e) Gripper.



(f) Reference grid.

Figure 8.1. Miscellaneous views of the synchronous dual-arm manipulation task.

object 90 degrees relative to the other to finally get the padlock to open. One last detail, we set the timeout of each simulation (i.e., episode) at 750 time steps, which corresponds to 18,750 time steps at the simulator level due to the frame skip. That is, in this proposed domain, an episode terminates only when such a timeout expires or when the task is fully solved.

Having presented a thorough description of our environment, now we will do the exercise of calculating in a fairly approximate manner the size of the smallest abstract state space that still contains a solution to the herein proposed task. We begin by identifying the maneuvers requiring the highest mechanical precision in this domain, which are the grasping of either object and the insertion of the key in the padlock. Figures 8.1(c), 8.1(d) and 8.1(e) shows relevant dimensions of the padlock, key and gripper, from which we can geometrically derive quite a few tolerances. For instance, the positional and angular clearances of a gripper as it clutches the key or padlock are (24mm, 1.57rad) or (10mm, 1.57rad), respectively. The linear tolerance is calculated along its slidding axis, simply as the maximum extension of the gripper minus the width of the given object. Meanwhile, we estimate the angular one around the lateral axis of the gripper. And for this, we just assume that this end-effector is perfectly aligned to grab an item by its width and that at worst its tips make contact with diagonally opposite edges of these prismatic items. Furthermore, it can easily be seen that the insertion operation has linear slacks of 3.92 and 3.52mm along two orthogonal axes parallel to the edges of the bottom face of the padlock. Plus, an angular tolerance of 0.155rad around an axis concentric to the shaft of the key. Clearly, any sensible abstraction of the observation space needs to take into account such clearances. Otherwise, if for instance we partition the 3D position space of the key into a grid that is too coarse (e.g., 5 cm per cell side), we might end up in the unwanted situation of having visited every cell but still not solved the task. Taking this into account together with the fact that a grapsing or insertion maneuver can be executed in any orientation. Then, it becomes reasonable to divide the Cartesian spaces of both end-effectors and both objects into grids of cubic cells whose sides are equal to their minimum positional tolerances. This means grid sizes of 10 and 3.52mm relative to the arms and objects, accordingly. At this point, to get a count of the cells in each lattice (i.e., macro states), we also must know the volumes of the workspaces corresponding to the end-effectors and the volumes reachable by the key and padlock. As a very gross approximation of the former, we consider that each workspace occupies one eighth of a sphere with radius of exactly the length of a fully extended arm (about 1 meter long). On the other hand, empirically we have seen both objects been thrown by our robot over 2 meters up in the air and over 9 meters away from the table. Hence, since it is rather hard to get more precise limits, we just assume here that the key and padlock are permitted to come anywhere within an empty cube of side length 3 meters. Based on these volumes, the total number of macro states originating from the previous four Cartesian positions is finally 5.2×10^7 per end-effector and 6.2×10^9 per object. We then apply the same reasoning to orientations, but in this case we take the range of each dimension to simply be between 0 and 2π rad. Using the only two angular tolerances mentioned before, we estimate 64 and 6.6×10^5 abstract states emerging from the 3D orientations of each end-effector and each object, respectively. If we now add up all macro states that we have acknowledged so far, we get a total of 1.8×10^{51},

a number that however contains plenty of rewarding states. To deduct the latter, we can make yet another not-too-outlandish assumption and say that for every abstract state of an object there is exactly one solution configuration involving the other 3 elements. A mental operation that in practice purely means that we should remove from the former tally all states associated with one object. Eventually, after doing so, we obtain the whopping figure of 4.5×10^{35} macro states per reward. And here we stop our analysis, since the previous number is so immense that it already gets our point across for the remainder of the document. Indeed, even without including velocities or the state of the gripper (i.e., open or close), this estimate is 28 orders of magnitude bigger than the count we estimated for Montezuma's revenge.

8.5 Benchmarking Setup

This section elaborates on our benchmarking effort, in which the previously described environment is confronted with three influential DRL algorithms proven to handle sparse-rewards.

8.5.1 Covered Algorithms

As usual, the algorithms evaluated in this benchmarking are the three listed in Subsection 3.3, namely: RND, ICM and ECO; plus PPO that primarily served as sanity check. We must also mention that, once again, the bonus formulation of ECO is altered, such that solely observations sent to its episodic memory provide any kind of intrinsic reinforcement. Here and now, it is worthy to contemplate what biases, if any, the three former methods have while exploring an environment (i.e., will they favor certain regions over others?). To begin with, ICM and ECO most definitely exhibit preferences by design, as they have been precisely develop to overlook specific kinds of distractors. ICM will ignore elements that are not directly controllable by the interacting agent, while ECO will dismiss distractors whose states are very few actions apart from each other. Crucially, none of these contraptions are present in the task proposed in Section 8.4; thus, further pondering over this is not relevant to this work. Apart from those two cases, none of these algorithms bear any explicit mechanism that would focus exploration, almost to the point of exclusivity, on certain parts of a domain in detriment of others. In fact, the spirit in which they would ideally solve a sparse-reward task is by meticulously visiting every nook and cranny of the world uniformly until finding a task reward. Only then, they will predispose their behavior to a noticeable extent and start reaching said reward more often. Still, these algorithms are not known for always working as expected, and in actuality, there are many situations that would cause them to derail from their neutrality. For instance, a situation widely regarded as negative, is when distractors different from those mentioned earlier trick these methods into becoming obsessed with them. In particular, distractors that are unpredictable or that can endlessly spurt never-before-seen states with little effort are known to trouble ICM and RND, respectively. And in the worst cases, such a fixation could get so overwhelming that such learners may entirely disregard the exploration of the rest of world.

8.5.2 Training Details

In terms of preprocessing, as personal choice, each observation is put together as a stack of 4 consecutive vector states, which are concatenated into a single vector with 240 dimensions.

Regarding their neural architectures, all methods employ a very similar policy and value networks. In the cases of PPO, ICM and ECO, the policy consists of a stack of 3 dense hidden layers with 512 units each and tanh activation, plus a linear output layer. Meanwhile, the policy of RND is also composed of 3 dense hidden layers with 512 units each and an output layer, but their activations are ReLU and tanh, respectively. Note that all these are stochastic policies; hence, their outputs represent means and variances from which actions are later sampled according to a Gaussian distribution. Concerning all 4 methods, their value networks have the exact same hidden architecture as their policies, but they are not connected to each other. Since PPO, ICM and ECO estimate a combination of intrinsic and extrinsic reward, its value predictor bears just one value head as output. In contrast, the value network of RND maintains two separate value heads, where each estimates one of the previous two reward sources. In all cases here, a head is a linear layer with precisely one neuron.

The random and predictor networks of RND use architectures that are identical to each other. Both have 3 dense hidden layers with 256 units each and leaky ReLU activation, which are followed by a linear output layer with 256 neurons. Importantly, these two networks receive as input solely the latest state from the 4-stack used by the policy. Likewise, EC concatenates the latest states of two different observations to form a 120D array, which is sent to its reachability classifier. Such a model is made of 4 dense hidden layers with 256 ReLU units each, which then connect to a softmax layer with 2 outputs. With respect to ICM, we only train a forward dynamics predictor and discard its inverse dynamics model. A decision taken for simplicity's sake, since the synchronous dual-arm manipulation task does not harbour distractors per se. Said predictor is fed with full 240D observations, which are first processed by a hidden ReLU layer with 512 neurons and then by a linear output layer of the same dimensionality as the inputs to this networks.

Table 8.1 provides a complete list of every other hyperparameter used in this benchmarking.

8.6 Results

We evaluated PPO, RND, ICM and ECO in our synchronous dual-arm manipulation environment. Each method was trained thrice, with a different seed each time, for 30 million time steps per run. In each run, we recorded a great amount of data; in particular, per simulation we gathered three valuable pieces of information. 1) The total reward collected, which as indicated in Section 8.4, episodically can only assume values of 1 (if the padlock is opened) and 0 (any other time). 2) The set of world sectors that have been visited at least once in the latest episode. Data that is independently taken in relation to each end-effector and object (i.e., key and padlock). Moreover, note that we merely kept account of whether or not a sector has been visited and not of how many times it may have been frequented during

Table 8.1. List of applied hyperparameters

Hyperparameter	PPO	RND	ICM	ECO
Number of workers	16	16	16	16
Rollout length	128	128	20	256
Number of optimization epochs	4	4	4	4
Learning rate	1.0e−4	1.0e−4	5.0e−5	1.0e−4
Entropy coefficient	0.01	0.01	0.01	0.01
PPO (λ)	1.0	1.0	1.0	0.95
Coefficient of extrinsic rewards	-	2.0	1.0	1.0
Coefficient of intrinsic rewards	-	1.0	0.01	0.03
Extrinsic discount factor (γ_E)	0.99	0.99	0.99	0.99
Intrinsic discount factor (γ_I)	-	0.99	-	-
RND experience fraction used by predictor	-	1.0	-	-
ICM curiosity loss strength (λ)	-	-	10	-
ICM forward inverse ratio (β)	-	-	0.2	-
ECO memory size	-	-	-	200
ECO action distance threshold	-	-	-	5
ECO negative sample multiplier	-	-	-	5
ECO reward shift (β)	-	-	-	0.0
ECO novelty threshold ($b_{novelty}$)	-	-	-	0.3
ECO aggregation function (F)	-	-	-	P_{90}
ECO reachability training learning rate	-	-	-	1.0e−3
ECO reachability training history size	-	-	-	120K

the same simulation. Ideally, for this monitoring, we would have liked to partition our manipulation domain in alignment with the analysis made in Section 8.4 (i.e., as many sectors as abstract states). But clearly, it would have been impossible to maintain a complete record of these visits due to memory constraints. Thus, instead, we just discretized the absolute 3D Cartesian space of this world into the 16x16x16 grid seen in Figure 8.1(f), and adopted its cells as the sectors to keep track of in all cases. 3) The salient events occurring within each simulation (and again, we did not log repeated executions during an episode). These comprised the following 8 primordial events: either end-effector grasping either object, either object losing contact with the table, both objects coming into contact with each other and the key being inserted in the padlock. Plus, 9 other composite events that are logically assembled from the previous ones. These included: either end-effector lifting either object (grasping + losing contact with the table), synchronous grasping and lifting of both objects, synchronous grasping or lifting with objects touching one another and synchronous lifting plus insertion. Based on all this data, the following graphs report several evolution curves. To clarify, these can be divided into two groups displaying: session-wise or episodic counts. Referring to the former lot, from the data collected during a single run, we simply calculate the cumulative count of some aggregate of visits or events over all episodes. As for the second group, there, at

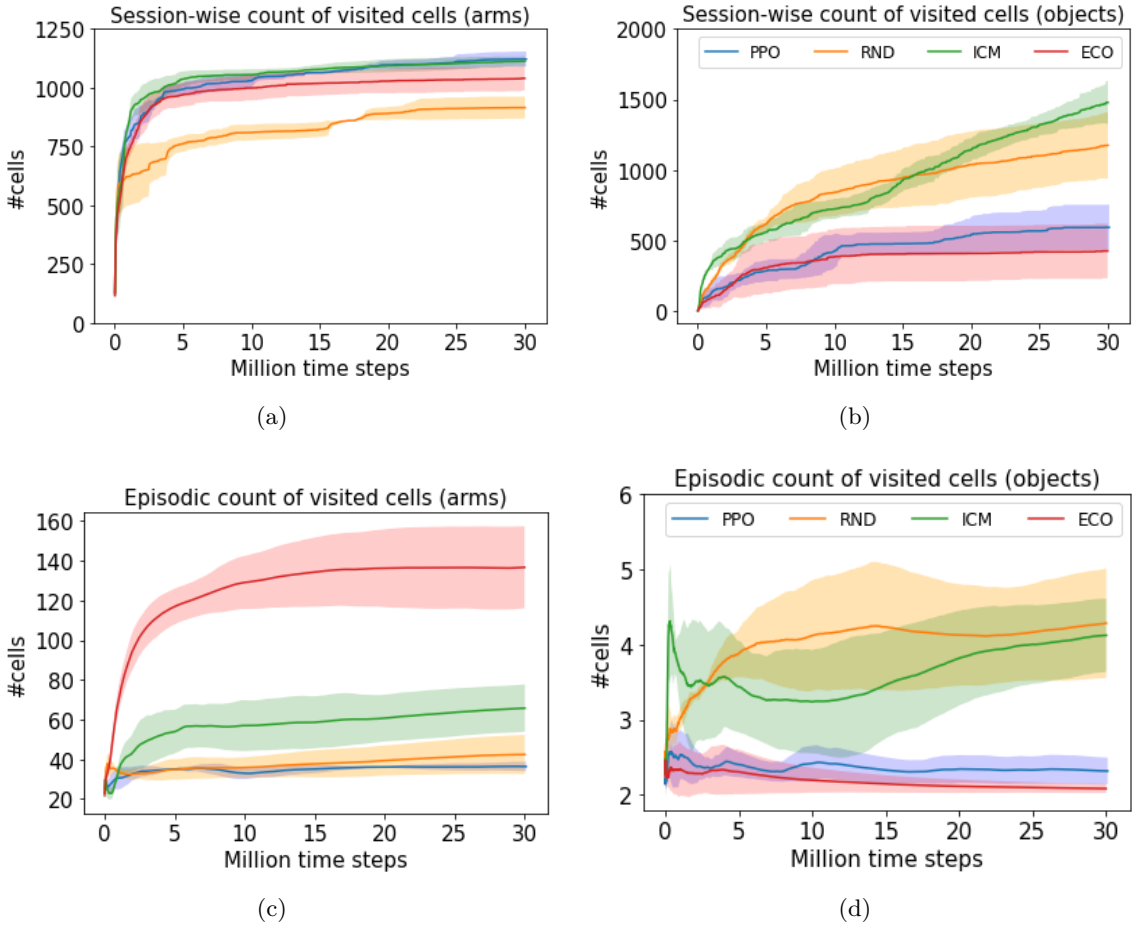
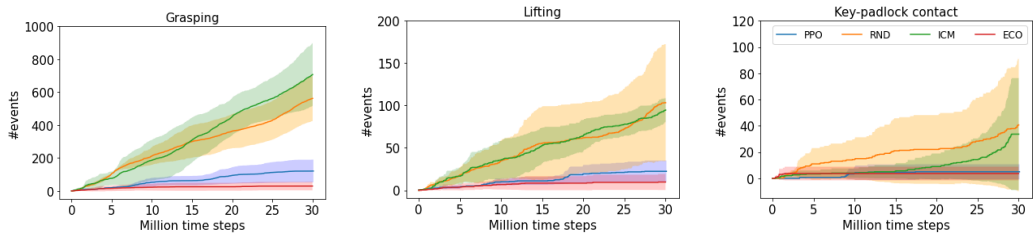


Figure 8.2. Visitation counts relative to both end-effectors and both objects.

each data point the same cumulative count as before is computed but now it is further divided by the number of episodes logged up to that point. Ultimately, these graphs depict the mean and variance across multiple training sessions of the previous cumulative results.

To begin with the analysis of the curves presented in Figures 8.2 and 8.3, we must point out that we have not rendered graphs corresponding to the task reward and other salient events strictly because in those cases the data is zero in every point of every run completed with each of the four algorithms. That said, the information visible here supplies two important results. First, as a sanity check, it strongly suggests that RND, ICM and ECO functioned properly in these experiments and it is unlikely that negative results are a product of external circumstances, such as a subpar hyperparameter tuning. In the cases of RND and ICM, we see that normally they explored much more of the world than PPO during a run, both in terms of frequented sectors and events. Specifically, after combining the visitations made by both end-effectors and both objects, we obtain that these techniques discovered 20 and 40% more cells respectively than PPO per training session. Not only that but if we concentrate on the counts of salient events the difference is abysmal. Clearly,



(a) (b) (c)

Figure 8.3. Counts of salient events.

only RND and ICM became interested in the key and padlock, at least enough to repeatedly grasp them, lift them and bringing them closer to each other. Regarding ECO, the main sign telling us it worked correctly is its high number of unique sectors visited per episode. In that respect, this method reported a count that is twice as large (sometimes even more) as those scored by the other three algorithms, a fact that aligns perfectly which the episodic nature of its exploration bonus.

The second, and obviously more important result, is that neither of the techniques being tested was even remotely close to solving the synchronous dual-arm manipulation task. As already exposed, they did not managed to open the padlock even once, and worse, by 30M times steps, none registered a simultaneous lifting event. Instead, what we see is that RND and ICM visited a great deal of the cells of our reference grids. As a matter of fact, by the end of a training session, they covered on average well above 25% of the cells making all four grids. Particularly interesting is the fact that these two algorithms explored more cells associated with the objects (mostly the key) than with the end-effectors. This intriguing outcome compelled us to investigate further by playing back episodes executed by such methods. By doing so, it became evident that they mainly enforce an explosive interaction with the key; i.e., they hit it and make it fly away. An interaction that, as mentioned earlier, is simulated chaotically by MuJoCo, which implies that the key will follow a very different path after each collision. Thus, seemingly, these learners are getting fixated on these always novel and hard-to-predict dynamics. In other words, exactly as speculated in Section 8.5.1, they might be acting as distractors. All in all, these are rather interesting results, because at first glance, that the robot is genuinely interested in the key and padlock should be celebrated. Nevertheless, it is questionable that RND or ICM will ever transform this raw attraction into complex behavior (e.g., grapsing), which due to being repetitive and predictable might not be as rewarding. Indeed, all curves in Figure 8.3 show a linear tendency, which given the cumulative nature of such graphs, indicate that the corresponding events occurred at a constant rate throughout a session. This lead us to believe that they may be happening by chance, as an unfortunate side-effect of trying to throw the objects off the table. And ultimately, there is no conscious effort over time to learn and execute more intricate maneuvers. As for ECO, we observe, as it is its nature, that it rapidly learned to follow with just the grippers a trajectory that does not curl on itself, i.e., that passes over many unique states in one episode. Then, because such a policy is optimal with respect to the bonuses distributed by ECO and will

forever remain that way, it tends to play the role of a local minimum preventing the emergence of other much different strategies. Certainly, the entropy regularizer of ECO is still expected to force some changes, but likely, they will commence toward the end of said optimal trajectory. Meaning that it might take a very long time before this method starts paying attention to the key and padlock.

8.7 Discussion

It is fairly clear from the previous section that RND, ICM and ECO cannot handle our robotics environment. So why is that? For us, the most fundamental answer is the excessive computational resources needed just for running an environment with 4.5×10^{35} abstract states per reward enough time to allow the former methods to reach a solution. As mentioned in Section 8.5.1, under a normal operation, these algorithms manifest no biases toward any part of a domain during exploration. Therefore, in the worst case scenario, they ought to visit every single macro state surrounding an incentive before discovering it. Unfortunately, with respect to our manipulation task, the number of interactions with the environment that must be realized before reaching that point is too large to be computed with current technology. Even if we greatly exaggerate and suppose that every single interaction leads to the uncovering of a brand new macro state, we will still require as many as the size of our abstract space. And if we further assume that simulating one interaction takes one nanosecond in a single thread of a processor, we will warrant around 10^{20} parallel threads just to be able to visit every macro state under a year. To put this into perspective, to date, the most powerful supercomputer in the world is Frontier, and it is just a bit shy of 18 million threads. Therefore, it will take a long time before we can run enough interaction steps so that RND, ICM or ECO could have a chance of working. Moreover, since this limitation has to do only with the execution of the environment, we additionally claim that a problem this large is at the moment intractable for any machine learning method. As long as they do not alter the original domain in any way through the introduction of human knowledge, and they likewise perform an unbiased exploration of the world.

Another relevant question is: can these on-the-surface simple robotics tasks, which in reality have extremely sparse rewards and sneakily gigantic exploration spaces, be solved by introducing expert knowledge? If humans already have a solution for the task at hand, then the answer is decidedly yes. Several broad machine learning methodologies can readily be brought to our rescue, including: learning from demonstrations, imitation learning, inverse reinforcement learning and advice-base learning. Reward engineering could also be applied, though it might require some trying trial and error in all but the most trivial of tasks. Obviously, all these alternatives will inherently change the reward structure of the task, it will no longer be sparse. From the literature on learning with sparse rewards, reverse curriculum learning [29] could potentially solve many of these problems. This method requires to alter the initial state, replacing it at the beginning of training by a state near the goal state. For example, in our manipulation environment, this would mean to set the Baxter robot to start an episode holding both objects with its grippers, and the key at least inserted in the padlock. Thus, this algorithm urge

us to manually complete the task once or to play around with the initial joint and pose configurations until finding a state in the vicinity of the solution. But what if humans do not have an answer to a problem. Then, an interesting idea that needs further research is to introduce subtle general biases that could transfer to large variety of robotics problems. A perfect point of reference are naturally humans, which ourselves have many biases. For example, we constantly exploit our ingrained understanding of basic physical transformations such as translations, rotations or reflections. After investigating a small object by rotating it in one of our hands, we avoid doing the same thing 20cm lower or to the left, as the outcome would be the same except for a translation. We also possess the notion of functional equivalencies. That is, we realize that after having inspected an object with our right hand, there is little value in repeating this very process with our left hand. Furthermore, we lack the insane obsession of wanting to observe every configuration of a body, including our own. Indeed, there are likely an unfathomable number of distinct joint configurations combining both our arms that though feasible we have not executed them in our lifetimes. Yet, exploring them is no longer a priority for most of us. Yet another preference exhibited by us is toward the interaction between objects. Infants naturally develop the motivation and skills for object construction (affixing, stacking, linking, etc.). If you give them a set of blocks, they are more likely to build towers with them than to systematically try every 2D arrangement over a carpet. By adopting all these biases and more during exploration, we expect new algorithms to sizeably reduce the exploration spaces they could be faced with. This, and still being able to solve any mechanical problem intended for humans, even those necessitating of high-precision maneuvers.

Chapter 9

Backtracking-Enhanced Sparse Episodic Curiosity Through Reachability

Episodic curiosity through reachability (EC) is a promising bonus-based algorithm proven to handle challenging sparse-reward environments. On top of that, EC has a few unique traits that give it an edge over other methods that also deal with reward sparsity. In particular, its bonus signal is grounded on a fixed property of the environment, which improves stability and hence sample efficiency. Moreover, EC outwits distractors such as a broken or a noisy TV; a feat that hardly any other approach have matched. However, and despite the previous benefits, EC still exhibits a few pressing drawbacks of its own. In light of this, the present work tackles one of them; namely, its inability to solve domains that are simultaneously sparse and deceitful. Here, a deceptive environment is one encompassing unrewarding goalless paths that nevertheless, by being the easiest to explore, lure EC into traveling to their inescapable dead-ends and to nowhere else in the world. To overcome this, we propose two modifications to this learner. Specifically, we counterintuitively replace its dense bonus function with a sparser one and, more significantly, we incorporate conducive backtracking bonuses associated with revisits to previous locations within an episode. It is these latter rewards that we expect will motivate EC to quickly back away from any encountered dead-end, enabling it to then move on to new and more fulfilling paths. Experiments carried out in a domain purposely designed to display the above problem evidence that our enhancement is absolutely essential for achieving success. Furthermore, we see no signs that our improvement has any negative side-effects on the desirable features EC already possesses.

9.1 A Flaw in Episodic Curiosity Through Reachability

Episodic curiosity through reachability [84] is a successful and rather interesting DRL algorithm that expressly addresses the problem of reward sparsity. EC is a bonus-based approach; in other words, it resolves the former issue by computing its own dense rewards, known as bonuses, which are parallel to those generated by an environment. Specifically, it awards real-valued bonuses that grow with the average

distance from the current observation to a selected group of others seen earlier in the same episode and whose computed bonuses were greater than a threshold. In such manner, these bonuses promote exploration and, ideally, policies that navigate lengthy trajectories without unnecessarily retracing their steps. To determine when two observations are close to or far from each other, EC relies on a reachability network trained to predict if two of them are, respectively, less or more than a predefined number of actions apart. As the above bonuses ultimately depend only on an unchanging property of the environment, i.e., the number of actions needed to travel between locations, they tend to smoothly reach a steady state. A feature that stands in stark contrast with other prominent bonus-based algorithms (e.g., RND [12] and ICM [74]), whose bonuses are heavily correlated with their policies and thus vary wildly during training. Such fluctuations move the learning target back and forth permanently; and as a result, both those methods show less stability and sample efficiency than EC. Another advantage of EC comes from its immunity to certain distractors such as a broken or a noisy TV that is always seen by an agent alongside the real task (refer to [84]). Since such distractors establish that every new TV station is to be drawn from a uniform distribution, any two of their states do not hold any information relative to the number of actions between them. Therefore, these TVs are irrelevant to the reachability network of EC, which over time will learn to ignore them from every observation and bonus computation. Notably, algorithms such as RND and ICM are known to become completely fixated on the same TVs. They wind up just staring at their never-ending streams of novel stations, while making no effort to solve the target task.

Notwithstanding the previous benefits, EC has its own problems. The one we focus on here is its propensity to fail in environments presenting deceptive paths, that is, ones that are easier to explore than those leading to terminal goals. Obviously, in such domains, these easier paths will be the first ones EC learns to fully traverse, as long as it works as intended. And at that point it will be under the impression that they are the longest around (even if the opposite is true). Furthermore, in problems with sparse rewards, we need EC to be innately interested in reaching further and further away from the start. Hence, we must configure it in such a way the cumulative sum of bonuses along a path increases with its perceived length. In consequence, since much like every other properly-tuned DRL algorithm EC is greatly driven to maximize reward, it will exhibit a bias in the early stages of training toward these easier branches that seem more rewarding. Not only that, but sufficiently large differences between cumulative sums of bonuses may trump the exploration drive of EC, causing it to visit the difficult paths too little to make further learning progress in them. Yet, the biggest issue is that once the latter deadlock materializes EC hardly ever breaks out of it. This happens owing to two reasons. On the one hand, seeing that its bonuses stay roughly constant as time goes by (e.g., they do not decrease with visitations), avoided paths never become appealing reward-wise. On the other, this algorithm was not explicitly designed to urge agents to double back on their trajectories within an episode once they reach a dead-end. Still, it can be configured to achieve this by specifically using a percentile score or a similar one to aggregate individual distances within the bonus computation. By doing so, bonuses are then positively related to the current size of the selected group of observations mentioned a paragraph ago. As a result, an

agent that always moves away from the start location will generally perceive that the further it goes the bigger such group grows and, in turn, the larger bonuses get on average per time step. Not only that but as the group grows so does the potential intrinsic reward that can be collected from previously seen observations, which tends to be greater nearer the start. Ultimately, it is this gradient of potential bonuses that in theory may give rise to a backtracking strategy after a dead-end is encountered. Sadly, such a bonus distribution produced by EC is too incipient to push an agent to retrace its steps in most cases. Among its many drawbacks; first, it is simply too unchecked to work in every domain, and as a matter of fact, we did not see it happening in our experiments. Besides, in very large environments, with group sizes clocking the thousands, an agent will experience a fairly flat bonus landscape for hundreds of steps, which is highly undesirable when dealing with sparse rewards. Moreover, when facing an extremely troubling section of the world, EC may choose to backtrack instead of trying harder to move ahead. Finally, if there is more than one deceptive path, this algorithm will prefer to cycle through them endlessly rather than exploring the difficult parts of the domain.

This work forwards one answer to the aforementioned problem, which consists in endowing EC with backtracking capabilities more advanced than those it already has. To do so, we first make the bonus sparser by defining that an agent receives a unit reward only at observations that are sufficiently far from every other that previously handed it an intrinsic reward in the present episode. In any other case, it receives a bonus of zero. We refer to these rewarding observations as milestones. Given this new formulation, we then distribute extra backtracking bonuses whenever the agent revisits a milestone, i.e. reaches an observation close to it after being afar. Importantly, we also add two hyperparameters that control the magnitude of such backtracking bonuses and the maximum number of revisits that are rewarded. Thanks to all these improvements, we expect our approach to be motivated to retrace its steps upon hitting the dead-end of some fruitless path and to do so without suffering any of the four drawbacks mentioned above. In this way, provided a large enough timeout per simulation, it should solve misleading domains by learning to cover every inch of them within a single episode. To test this refinement, we crafted a simple environment that showcases sparse rewards together with deceptive paths. Experiments carried out in this scenario reveal that EC misses to complete the task every time. It invariably converges to a policy that nearly always takes an easy-to-explore yet goalless path, making very little progress toward the actual goal throughout a training session. Meanwhile, our backtracking-equipped proposal solves the entire problem almost effortlessly, displaying a fast and steady convergence rate. We additionally run an experiment where we placed a distractor in the shape of a noisy TV on top of the previous domain. Also in this case our method proves to be highly competent; it handles such a TV distractor just as well as EC, showing no signs of detrimental side-effects.

9.2 Episodic Curiosity Through Reachability

EC is a DRL algorithm designed to deal with sparse rewards and with certain distractors (i.e., dynamic elements with no relevance to the solution of the task at

hand). It follows a bonus-based approach; that is, at each time step t, additional to the rewards handed out by the environment, known as extrinsic rewards r_t, it offers interacting agents reward bonuses b_t, also called intrinsic rewards. In particular, the conceptual scheme obeyed by EC is to grant bonuses that are positively correlated with some average distance between the current observation and a set of past ones that themselves were highly rewarded in the same episode. This means that observations which are far from most previous ones along an uninterrupted trajectory will yield larger bonuses. Furthermore, a correct implementation of EC requires first and foremost that the cumulative bonus that can be collected from a given path in the environment grows with its known length. In this way, this method overcomes sparse-reward settings by being internally driven to travel increasingly longer distances in the hope of eventually finding extrinsic rewards.

From a practical standpoint, the implementation of EC requires two ingredients. First, a reachability network R responsible for predicting if any two observations are less or more than k actions apart, where k is a user-specified hyperparameter. For convenience, R is further divided into two modules: embedding E and comparator C networks. The former receives two observations, o_i and o_j, processes them independently and produces two embedding vectors, e_i and e_j, of a much lower dimensionality than the starting inputs. Meanwhile, the comparator network takes both these vectors and outputs a reachability prediction: $c_{ij} = C(e_i, e_j) = C(E(o_i), E(o_j)) = R(o_i, o_j)$. Such prediction is a scalar value between 0 and 1, where lower means further apart. The second ingredient is an episodic memory buffer M that stores key embeddings, which in this work we refer to as milestones. This buffer is additionally employed to compute a similarity score S, which is simply an aggregation of the reachability predictions between some target observation and all the ones implicitly found in memory: $S_F(o) = F(\{C(E(o), e_m), \forall e_m \in M\})$. Here, F is a suitable summary statistic; common choices include the maximum or the 90th percentile. Based on the previous two components, EC operates as follows. At the beginning of every episode, M is emptied and then loaded with just the embedding relative to first observation generated by the environment. Afterwards, at each subsequent time step two things happen. The similarity score associated with the current observation is evaluated and if it is smaller than a predefined threshold S_{far}, its corresponding embedding is added to M. A less abstract way to think of this is: if an observation is far from every milestone, in terms of number of actions, only then it is appended to M. Plus, a bonus is awarded to the interacting agent; computed as in Equation 9.1, where α and β are two hyperparameters that determine the scale and sign of the bonus, respectively.

$$b_t = \alpha \cdot (0.5 - S_F(o_t) + \beta) \tag{9.1}$$

$$\hat{r_t} = r_t + b_t \tag{9.2}$$

Regarding its training procedure, EC seeks to learn two models: a reachability network and an action policy, both formulated as neural networks in the original paper and likewise in this study. They can be learned simultaneously or sequentially (the reachability model before the policy). Moreover, their training is conducted independently; in other words, for each model there is a separate optimizer and separate training hyperparameters, including: learning rate and update frequency,

among others. Going into detail, the training of the reachability network is framed as a binary classification problem and carried out in a supervised online fashion. Particularly, at each update, EC samples an equal number of positive and negative examples from a buffer that contains all transitions experienced since the last iteration. Examples that are each composed of two observations appearing less (positive) or more (negative) than k time steps apart along the same recorded episode. Note that the source data supporting this undertaking are trajectories executed by a uniform policy (sequential mode) or by the action policy being concurrently tuned (simultaneous mode). As for learning a reward-maximizing policy, EC internally delegates this job to the well-known general-purpose DRL algorithm PPO [88]. Critically, PPO is asked by EC to optimize an augmented reward signal $\hat{r}$ that combines extrinsic and intrinsic rewards, as shown in Equation 9.2.

As pointed out by its authors EC has a few advantages over similar methods. For instance, when they tested it against ICM in a visual navigation task (without distractors), EC was reported to be twice as sample-efficient. A likely explanation involves the glaring differences in how the bonuses generated by these algorithms evolve in the long term. In regard to EC, clearly, when learning is done sequentially, bonuses are static. But even in the simultaneous mode, over time its bonus signal tends to converge smoothly to a fixed landscape as a result of being directly related to an immutable property of the environment (i.e., number of actions between observations). Thus, in all cases, it constantly offers policies a stable scenario to learn in. On the other hand, the bonus landscapes produced by the likes of ICM and RND report sizable time-dependent oscillations in the locations of their peaks and valleys. The main culprit of such variability is the correlation between policies and bonuses enforced by these two methods. A policy is attracted to regions possessing large bonuses, but by reaching them repeatedly it causes just those bonuses to drop. Yet, other factors also come into play; such as catastrophic forgetting, which causes some bonuses to change unexpectedly. Sadly, these oscillations are not always favorable; and in fact, they often drive policies toward already explored parts of an environment, wasting valuable resources in the process.

A second benefit of EC resides in its ability to ignore certain distractors. Specifically, those whose observations do not provide enough information to determine the number of time steps needed to go from one to another. One example of such is a noisy TV. This distractor is only applicable to environments with visual observations as it attaches an image, called station, to each of them. The placement of the TV screen must be such that what can still be seen of the actual task in every observation is sufficient to solve it (note that if this is not true then EC might run into problems). For instance, with respect to an observation image from the original environment, the TV could: be put on top of it (if it is small enough), be stitch next to it or occupy new channels, among other options. In addition, the interacting agent is given one extra action to change stations. And when it does, a new image is selected uniformly at random, either from a database or in compliance with some generative process. This noisy TV was experimented with over a navigation task by the authors of EC, who tested their own algorithm and ICM. Their findings demonstrate the superiority of EC in this particular setting. While ICM usually gets obsessed with the TV and stops exploring much of the real world, EC pays no attention to this distractor and completes the underlying task. The reason seems

obvious; seeing this TV does not improve the accuracy of a reachability network, as it is impossible to grasp how many actions apart two stations are. Therefore, such a network is inclined to dismiss the TV entirely from its predictions, prompting EC to hand out a minuscule bonus whenever an agent changes stations.

9.3 Deceptive Bonuses Created by EC

As said in the previous section, EC relies on PPO to learn a policy that maximizes total extrinsic plus intrinsic reward. PPO is an on-policy DRL method that works by training a value and a stochastic policy network, which is accomplished thanks to two objectives. A value function error term whose reduction increases the accuracy of the value network. And a surrogate policy gradient objective whose minimization maximizes the expected cumulative discounted reward collected by the policy. Optionally, a third objective might be incorporated; an entropy bonus whose diminishment compels the policy to choose actions more uniformly. These orthogonal objectives are finally minimized together with one stochastic gradient optimizer after being aggregated into a single score through a weighted sum. Unfortunately, it is well-know that PPO misbehaves in deceptive environments where the path leading to the solution appears at one point to hold less reward than other areas. Clearly, in this scenario, its policy gradient objective would wrongly urge PPO to reach those areas and to utterly disregard the optimal path indefinitely. In theory, elevating the weighting coefficient linked to the entropy bonus should alleviate the previous situation, but in reality this is not always true. Case in point, long-horizon problems put an upper limit on the relative importance of entropy over reward maximization, since too much of the former would hinder an agent from traveling very far from its start location. It is worth pointing out that this bias toward bigger rewards is not exclusive of PPO. In fact, most, if not all, general-purpose DRL algorithms exhibit it too, including: DQN [65], DDPG [60] and SAC [38].

On certain sparse-reward environments, EC likewise suffers from the above problem, which in its case paradoxically arises as a consequence of its own bonuses. Consider the environment in Fig 9.1, where from its start position an agent can choose one of two branches. It can either take the path toward the left room, a dead-end, or toward the right room, which culminates in the goal. Crucially, the left branch has no deadly states while the right one is riddle with them, making it harder to traverse. Intuitively, such a disparity in difficulty encourages EC to finish the exploration of the left room first, and by a large margin. Giving it to believe at around this point that the left path is the longest by far out of the two. In the end, it is because of this deception that EC develops a visible and unwanted bias toward the left branch since, as we have already explained, when this algorithm operates properly, it accommodates greater cumulative bonuses in longer paths. Exactly how pronounced this bias will be is not trivial to determine, as that depends on multiple factors. But one thing is sure, if it is intense enough such that the right room is too sporadically visited, then EC will have very little hope of solving this problem. First off, the bonus landscape is simply not expected to shift in the foreseeable future, which makes the right room perpetually unattractive to EC. This is true by definition when learning in sequential mode. Yet, it happens all the same in the

simultaneous mode, because low visitations imply not enough data about this deadly room to make a difference in the loss computation of the reachability model. Besides, EC does not wield any mechanism that would spontaneously change its intrinsic reward signal while its policy remains essentially static (i.e. bonuses do not decay with time or revisits). One last option for EC to still find success in the described problem would be if it manages to double back along the left path after reaching its dead-end. And in fact, as reported in Section 9.1, there is one configuration that allows this; namely, when the aggregation function F is set to evaluate a percentile. Because of the nature of this statistic, an interacting agent can obtain several large bonuses at the same milestone without even moving instead of just one; the latter being the case for other scores such as taking the maximum. Crucially, the precise number of such effortless bonuses that are retrievable per milestone is related to the current size of the episodic memory and ultimately to how much of the world has been covered up to the present time step. Thus, while traveling straight through a long path, the agent will receive not more than one large reward at milestones neighboring the start location, but it may get tens per milestone near the end of the road. At this point, since such limit applies to all milestones concurrently, past ones would harbor a significant amount of grand bonuses, which could be plenty to give rise to a backtracking behavior. Sadly, as indicated earlier, this strategy is too rudimentary and, as such, it exhibits numerous drawbacks and it is heavily restricted in its applicability. At best, EC could tackle a domain with strictly one misleading path, but if there is even a second one, it will proceed to cycle through them and to ignore everywhere else. After each time the agent completes a round trip along one path, the above-mentioned limit will have raised considerably, making all others as appealing as the first time they were traversed, if not more.

9.4 Enhancing EC with Sparse and Backtracking Bonuses

To tackle a sparse and deceitful environment like the one examined in the previous section, we elaborate on an idea already expose above. That is, we accept that our algorithm will be lured by misleading paths, but we endow it with an explicit bonus signal that will guide it to backtrack from their dead-ends to their originating bifurcations. Plus, contrarily to what EC does, we enforce a tighter control over these supplementary bonuses that ensures they will be well-behaved and work as intended in any domain. In the long run, our approach should learn an overarching policy skilled enough to travel across its entire world in one sitting; a strategy we reason must discover every extrinsic reward in various scenarios.

We construct our method by taking EC as template, upon which we impose a few adaptations. First of all, we make the bonus function sparser, which is to say, many and indeed most observations will be assigned a reward of zero. This is accomplished by replacing the original bonus formula with Equation 9.3, where $\mathbb{1}$ is the indicator function and S_{max} denotes that we restrict F to be no other statistic but the maximum. Therefore, this variant hands out a unit bonus (later multiplied by the scale term α) when the current observation is objectively far from every milestone (hence, it is also recognized as a milestone itself) and zero otherwise. Another way to visualize this is that the former rule divides an environment into

numerous equal-sized (depends on S_{far}) overlapping regions, each centered at a milestone. And it reinforces agents exclusively at every first individual visit to any of these. We decided to use a non-dense bonus signal primarily for practical purposes; as this facilitates later on the definition of coherent bonuses designed to encourage a backtracking behavior. Regardless, it is worth briefly mentioning that empirically, having run a vast number of experiments across multiple domains, we have noticed that this tweak alone creates an algorithm notoriously less sensitive to hyperparameters than EC. An unexpected benefit that is much welcomed.

$$b_t = \alpha \cdot \mathbb{1}\{S_{max}(o_t) < S_{far}\} \tag{9.3}$$

As a second and more decisive modification, starting from the previous variation, we drop its constraint of granting rewards only on first arrivals and instruct it to give extra bonuses on follow-up visits to the same region (i.e., revisits). We name these supplementary rewards: backtracking bonuses. We implement this general concept with minimal computational effort by reusing the tedious calculations already performed at each time step by EC. Specifically, to calculate such bonuses, we make two crucial changes to the sparse version of EC described above. We install a mechanism for counting revisits; and, we introduce them into the bonus estimation. To identify follow-up visits, the rough idea is to associate each detection with a particular milestone stored in the episodic memory M. Then, at any given instant, a revisit is said to take place if the next two conditions are met. The current location of an agent is, in terms of number of actions, close to precisely one milestone while far from all the rest. And, such nearby milestone is not the last one frequented by the agent, either for the first or the nth time. Note that the latter provision ensures that to revisit a milestone an agent must priorly move far away from it. To evaluate the first requirement, we need a new hyperparameter. Another similarity threshold S_{close} that, analogous to S_{far}, help us determine when two observations are close to each other, i.e. when their reachability prediction is greater than this threshold. Whereas, the second condition requires the addition of a variable i_{last} that keeps track of which milestone has been visited most recently (by pointing to its index within M). This is updated whenever the episodic memory grows or a revisit is detected. Moreover, since we ultimately want to clock the number of follow-up visits, we require an additional array of counters N, which maintains a separate tally for every embedding in M. With these elements in place, the algorithmic process to count how many times an agent returns to some location goes as follows. At the start of an episode, N is emptied. Afterwards, at each time step, we repeat the steps of EC in which it initially computes a list of predictions $P(o) = \{C(E(o), e_m), \forall e_m \in M\}$ and later uses it to check if an observation o is a milestone, $max(P(o)) < S_{far}$. When the former inequality is true, concurrently to appending $E(o)$ to M, we update i_{last} to $|M| - 1$ and instantiate a counter linked to this latest milestone (initialized with a value of 1). On the other hand, when the same inequality is false, we repurpose the previous list to verify if at least one milestone is close to o, i.e., its maximum value is above the corresponding threshold, $max(P(o)) > S_{close}$. Then, provided the preceding clause is satisfied, we acknowledge that o denotes a revisit if it is further confirmed that: the nearest milestone has not been frequented lately, $i_{max} = argmax(P(o)) \neq i_{last}$, and the second largest value in $P(o)$ is below S_{far}.

Finally, following a detection, and before the next time step commences, we update i_{last} to i_{max} and increment $N(i_{last})$ by one.

Having calculated these counts, we must next decide how to construct a favorable intrinsic reward function that takes advantage of them. Since we are already working with sparse bonuses, we choose to stay on this track and design backtracking bonuses of the same nature. Following through with this decision, we equip our algorithm with the most straightforward assignment rule: it puts forth backtracking bonuses solely when revisits are detected. Furthermore, with regard to their magnitude, the simplest scheme would be to hand out a unit bonus on every follow-up visit. However, that would originate a relatively flat bonus landscape, which in turn may instill several unwelcome behaviors in a policy. Such as: iterating between two milestones to reap rewards while doing the bare minimum or preferring to take a previously traversed path over an unexplored one at a revisited junction. Therefore, we opted for having bonuses that grow smaller with each revisit; an obvious answer that nevertheless is anticipated to solve all the above problems. Indeed, diminishing bonuses should offer little reason to linger in the vicinity of a handful of milestones when larger rewards lie just beyond them. By the same token, uncharted regions should convey more tempting bonuses than their antitheses. In concrete terms, we formulate this bonus as a coefficientless monomial function, which takes as input the reciprocal of the count corresponding to the milestone that has just been visited, $N(i_{last})$. In this way, we have one more hyperparameter; an exponent δ that confers certain control on the decay rate of this reward. For extra flexibility, we also set up an adjustable cutoff N_{max} such that any visit count greater than this upper limit results in a bonus of zero. Our prevision here is that a lower cutoff, if still effective, may improve sample efficiency to some degree. Putting it all together, Equation 9.4 shows the bonus given by our fully-fledged method to an observation o_t that is deemed a revisit (note that this calculation is done after updating i_{last}).

$$b_t = \alpha \cdot \frac{1}{N(i_{last})^{\delta}} \mathbb{1}\{N(i_{last}) \leq N_{max}\} \tag{9.4}$$

Given that our solution relies on sparse bonuses as well as backtracking bonuses, in the remainder of this paper we will refer to it as backtracking-enhanced sparse EC, or BESEC for short.

9.5 Experimental Setup

This section details experiments realized to evaluate the performance of BESEC.

9.5.1 Environments

As part of this study, we have designed two domains that bring out into the open a specific problematic experienced by EC. Namely, its irrevocable bias toward easy-to-explore regions of an environment that regrettably do not play a role in its solution. In line with this, our first scenario (see Figure 9.1), right at the outset, places an agent (white) at a bifurcation with two branches. Of them, just the right one is peppered with numerous deadly states, which intuitively make learning to travel this branch thoroughly much harder in comparison to the left one. Notwithstanding,

Figure 9.1. Maze navigation domain. Views of all three rooms at the initial state.

the agent must traverse the right path to arrive at the singular terminal goal of this environment. In more detail, this is a static maze navigation domain, which is solved once the agent collects the burgundy jewel located on the upper rightmost corner of this environment. Such jewel represents a terminal state and it also provides a reward of +1 (no other reinforcement is featured in this problem). Critically, as it navigates, the agent must be cautious not to land on a deadly state (red), because otherwise the current episode ends at that instant. And it needs to be quick too, since every episode unconditionally terminates when its timeout of 500 interaction steps elapses. Regarding the maze layout, we have constructed it as a gridworld, divided into 3 contiguous rooms, each partitioned into 11x11 cells. These rooms are additionally split into different color sectors (i.e., arbitrary groupings of adjacent cells). Moving on to its perception, the agent has only partial observability of its world. That is, it solely sees the room presently enclosing it, with each observation being a 44x44 RGB image (visually identical to those shown here). At last, the action space consists of 4 discrete actions denoting four directions of movement: right, left, down and up. These actions always relocate the agent to a neighboring cell unless the latter corresponds to an impenetrable wall (black), in which case no motion occurs. However, transitions are not deterministic due to our implementation of sticky actions. In other words, with probability 0.25 an action chosen by the agent is superseded by the one performed one time step ago.

For our second environment, we simply introduce a noisy TV distractor within the previous domain (see Figure 9.2). Notably, to implement this distractor, we essentially replicate one of the setups used by the authors of EC. As such, we will be testing this method in the kind of harsh problem for which it was conceived. In terms of observations, we built each of them by taking the 44x44 RGB image generated by the maze simulator and placing it on top of a 88x88 color TV (i.e., every observation has dimensions 88x88x3). Moreover, this TV is partitioned into a 11x11 grid, where each cell is assigned one out of 16 colors. With respect to its actuation, the interacting agent is endowed with one extra discrete action that allows it to change stations (making a total of 5). Every time such action is executed, a new station is created by independently drawing colors uniformly at random for each TV cell.

Figure 9.2. Maze navigation domain alongside a noisy TV (diverse stations).

9.5.2 Training Details

In almost all our experiments, irrespective of the algorithm (EC or BESEC) or training mode (sequential or simultaneous) being used, we have employed the same neural architectures. The one instance where this is untrue is when running tests on the domain comprising a TV distractor. In that case, we maintained the general structure and most hyperparameters, but we were forced to increase the overall size of each network. Starting with the policy model, it is constructed as a two-head policy-value network consisting of two distinguishable modules: convolutional (receives images) and recurrent (receives embeddings), connected sequentially. The convolutional one is designed as a stack of 4 layers with 32 filters each, kernel size of 3, stride of 2, same padding and ReLU activation, followed by a flattening layer. This arrangement is replicated in experiments without and with a distractor, producing at its output embeddings of size 288 and 1152, respectively. In turn, the recurrent module is formed by a single classically implemented LSTM layer that concatenates in parallel with two output linear layers, corresponding to the policy and value heads. Moreover, the size of the LSTM is set to 512 or 1024 units, with the latter number being applied when dealing with a TV distractor. Regarding the reachability model, it is built as a siamese network where each of its two branches has an exact replica of the convolutional module used by its associated policy. These branches are followed by a stack of 4 dense hidden layers with 512 ReLU units each and, subsequently, by a softmax layer with 2 neurons. At last, of such an output, only its second dimension is taken as a measure of reachability.

As for hyperparameters beyond those involved in the configuration of the neural architectures, we must disclose that we did not change any of their values between training modes (sequential or simultaneous). Nevertheless, when executing the sequential mode, in an attempt to simplify learning, we certainly altered the operation of the evaluated environment during the training of reachability network. Specifically, while not modifying anything at the visual level, we made deadly states to act as impassable walls and the jewel ungrabbable. That way the only termination condition is the timeout, which we additionally increased to 2000 time steps. One more thing, we extended the support of the initial state distribution, such that this time the interacting agent can start an episode in any traversable cell of any room. All in all, these changes make it easier for a uniform policy to travel the world from end to end within a single simulation. Besides, they avoid a potential data imbalance

Table 9.1. Overview of applied hyperparameters

Hyperparameter	EC	BESEC
Number of workers	16	16
Rollout length	256	256
Number of optimization epochs	4	4
Learning rate	2.5e−4	2.5e−4
Entropy coefficient	0.1	0.1
PPO (λ)	0.95	0.95
Coefficient of extrinsic rewards	1.0	1.0
Coefficient of intrinsic rewards (α)	0.03	0.03
Discount factor (γ)	0.99	0.99
Episodic memory size ($\|M\|$)	500	500
Action distance threshold	5	5
Negative sample multiplier	5	5
Reward shift (β)	0.5	0.0
Dissimilarity threshold (S_{far})	0.5	0.3
Similarity threshold (S_{close})	-	0.9
Aggregation function (F)	P_{90}	max
Reachability training learning rate	1.0e−3	1.0e−3
Reachability training history size	128K	128K
Backtracking exponent (δ)	-	1.0
Backtracking cutoff (N_{max})	-	2

resulting from certain areas of the maze being more frequented than others. Finally, Table 9.1 gathers the remaining hyperparameters used in this study.

9.6 Results

We begin by comparing EC and BESEC in the simplest of our proposed maze environments (i.e. without a TV). In this experiment, we trained both methods in sequential mode. To this end and to ensure we evaluate them on an equal footing, the procedure we followed was to first train a reachability model during 20M time steps. Then, we used the same frozen network to learn a policy with each algorithm for a maximum of 50M time steps. In the end, for the sake of reproducibility, we repeated these procedure thrice per network, with a distinct seed each time (for a total of 9 seeds). Additionally, to validate the correct operation of EC, we preliminary tested this approach on a domain where navigation in our maze is limited to just the right branch. And for that, we executed three runs, each employing a different reachability network from the ones that we had pre-trained.

Figure 9.3 reports the performances of EC and BESEC in the experiments described above. Note that, each curve is obtained by first computing a moving average with a window of 250K time steps over an entire run and afterwards taking the mean and variance over three runs. Moreover, relative to the left branch, our

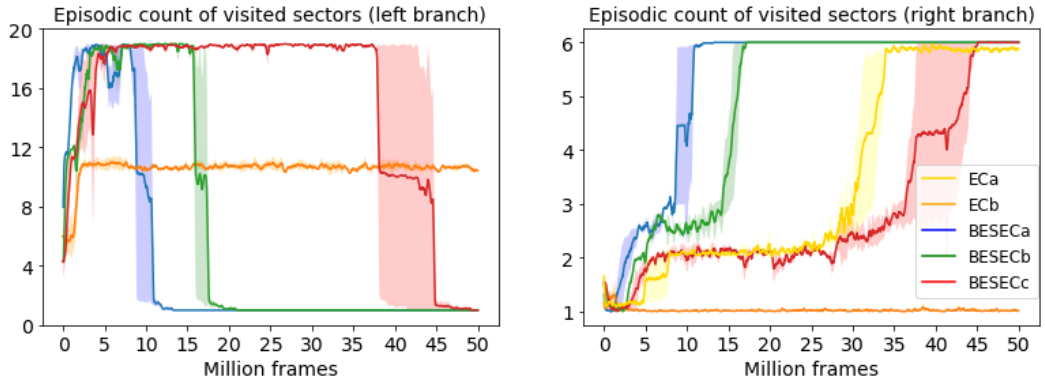


Figure 9.3. Results from various experiments. Identifiers in the legend correspond to the following experiments (clearly, the name of the algorithm that was tested is already written in the identifier). ECa: Only right branch - No TV - Sequential mode. ECb: Full maze - No TV - Sequential mode. BESECa: Full maze - No TV - Sequential mode. BESECb: Full maze - No TV - Simultaneous mode. BESECc: Full maze - TV is present - Sequential mode.

count considers the number of sectors an optimal agent passes through during a round trip from the start location and back (revisited sectors are tallied twice), thus, its maximum is 19. Meanwhile, for the right branch, we consider only the sectors visited on a one way trip from start to goal, meaning that its corresponding count can be at most 6. These results consistently reveal that when an agent is to navigate exclusively within the right branch, EC is perfectly able to find the goal every time before the predefined budget of 50M time steps expires. Nevertheless, when this algorithm is faced with our deceitful maze in full, its inability to solve it becomes quite clear. We observe that it normally reaches over 9 sectors along the left path and not many more than one over the right one. This is a strong evidence that our analysis in Section 9.3 is correct, indicating that EC develops a visible bias toward deceptive paths. Furthermore, its behavior after convergence remains fairly stable; suggesting, as we rationalized earlier, that the local minimum represented by the left path is virtually inescapable. It should be noticed as well that this method visited approximately 11 sectors of the left branch per episode. This tells us that at least in this domain the theoretical mechanism that should allow it to backtrack is not working, even when percentile is being employed as aggregation function. On the other hand, BESEC swimmingly handles our tricky scenario. It learns in about 10M time steps and does not seem to have had a hard time at any point. In fact, confirming the sensible formulation of our bonus signal, it swiftly proceeds to acquire a backtracking strategy as soon as it masters to consistently arrive at the dead-end of the left branch.

To investigate how versatile BESEC is, we carried out two more experiments. In one, we trained our algorithm in simultaneous mode, still in the simpler maze; besides, as mentioned before we kept all hyperparameters unchanged from when we run it in sequential mode. In the other, we evaluated this method in our second domain, which contains a noisy TV; and in this case, we once again executed it in sequential mode. For these two trials, we reiterated the considerations described

in the previous paragraph, which is to say, each experiment was repeated on three occasions with varying seeds. Likewise, each run lasted 50M time steps when training the policy network, while in the latter experiment the reachability model was trained for 20M time steps. Fig 9.3 further reveals that in all cases BESEC retains its effectiveness. In simultaneous mode, it converges to the goal in less than 20M time steps and only stalls a bit while learning its way through the entrance sectors of the rightmost room. Yet, it ends up being merely slightly slower than when running it in sequential mode. As for its behavior in the presence of a TV distractor, BESEC still manages to come up with a proper solution. However, here we detect a noticeable slow down as it tries to overcome the cyan sector of the right branch, which holds the largest number of deadly states. Seeing that our algorithm nonetheless doubles back on the left path as fast as usual, we posit that such nuisance is inherent to EC. Our hypothesis is that since a reachability network never attains a perfect accuracy, an agent could get a few unexpected rewards just by persistently changing the TV station in place. And to an incipient on policy learner, that course of action may temporarily seem more rewarding than trying to go across a treacherous mined field, at least until its navigation skills improve.

Chapter 10

Conclusions

In this thesis, we have investigated from multiple angles the complex issue of applying deep reinforcement learning in sparse-reward domains. To begin with, we have elaborated an overview of environmental features that to this date impact state-of-the-art methods. Then, we have conducted four benchmarking studies, where each targeted an open problem involving a not yet well-understood environmental property. Lastly, we have designed a new algorithm that displays a visible improvement with respect to its immediate predecessor. Next, we write down all the conclusions derived from the previous research efforts, presented on a chapter-by-chapter basis.

- In Chapter 4, we have consolidated a thoughtful review of numerous environmental features that have a heavy impact on deep reinforcement learning in sparse-reward environments. In total, we have covered nine features; two, resettability and reversibility, that have been exploited by previous works and other seven that have been reported as demanding for one or more families of algorithms with advanced exploration strategies. For each case, we have delivered clear definitions and useful example environments. In addition, we have summarized experimental results and discussions found in previous studies that shine a light on which algorithms and their components are affected by a certain feature and what mechanisms underlie this interplay. We hope the benefits of this compilation effort to be twofold. First, the bulk of information contained here could assist researchers interested in analyzing the generality of their novel proposals. They could do this by validating from a theoretical perspective if their formulations share the same specificities or weaknesses as previous ones or if similar mechanisms take place in them. Taking this a step further, we would also like to see such generality assessment to be carried out empirically in a set of benchmark environments, where each one of them is intentionally crafted to highlight exactly one of the features herein reported. And second, through its presentation of challenging environmental features, this work points researchers towards various open problems involving sparse rewards. Some impact only a reduced group of algorithms (e.g., action-prediction degeneracy), while others reveal a more ubiquitous problematic that enwraps approaches from diverse lines of research. And within this latter category, we regard the phenomena of retracing steps, distractors and procedural generation as the most fundamental and stimulating challenges of the lot, whose resolution

would lead to the next generation of exploration strategies.

- In Chapter 5, we have proposed two new sparse-reward environments, each displaying an exploration-intensive distractor modeled as a TV. In addition, we have performed a benchmarking that tested three relevant algorithms on said environments. Findings from this benchmarking show that our tasks were challenging enough for these methods. In particular, all of them remained permanently mesmerized by the existence of a TV in at least one of our environments. And as a consequence, their exploration of the main task was severely hindered, to the point that they were no longer able to solve it. All of this leads us to conclude that current algorithms, by and large, are not yet sufficiently competent to handle exploration-intensive distractors in a comprehensive and principled manner. Finally, by making these environments open-source, we hope that they serve as a helpful reference for advancing the research in this topic.

- In Chapter 6, we have studied the response of state-of-the-art algorithms to sparse-reward environments demanding their re-exploration within the ongoing episode. To this effect, we have fabricated two novel tasks following the key-locked door paradigm, which requires an interacting agent to travel from the door to the key and back while visiting twice every location in between. In these domains, we then evaluated the performance of three renown techniques: RND, ICM and ECO. These experiments finally report that all these methods manifest virtually zero interest in retracing their steps; therefore, they have no hope of completing our tasks. Our ensuing analysis puts the finger on their exploration apparatus, materialized in their bonuses, as the culprit of this incompetence. Because they make decisions based almost entirely on current observations, they fail to accurately internalized critical past events that are non-longer visible (e.g., the collection of the key). Indeed, in the formulations of RND and ICM, we observe no mechanisms in place that would grant them the retentiveness necessary to foster any backtracking behavior. Contrarily, ECO does have an episodic memory buffer, but this is not used adequately for the purpose of a more conscious re-exploration that is triggered by a salient event. In time to come, as one potential solution to this issue, we would like to see more research in ways to smoothly and efficiently integrate memory atop the algorithms analyzed here. In such scenario, their bonuses would compute novelty, curiosity or reachability us usual, but over a mostly imagined complete picture of the world instead of over partial observations.

- In Chapter 7, we have put forward a new sparse-reward environment, the D2K2 domain, in which it is easy to detect the occurrence of skill reuse and quantify its effects on sample efficiency. This allowed us to straightforwardly define an informative RSSE indicator that gives a good contextualization of how sample-efficient an algorithm is and facilitates comparisons among different learners. We believe this environment is an ideal lightweight benchmark for evaluating and tracking improvements in sample efficiency, with focus on skill reuse and efficient exploration under sparse rewards. As part of this study, we also carried out a first benchmarking in such domain involving

three high-performing algorithms known to handle environments with very sparse rewards. We report that none of these methods gave any indication of being sample-efficient. None achieved a mean RSSE higher than 0.33, i.e. to generalize a skill they all demanded more than three times the samples required to discover it. Ultimately, the numerical scores obtained here provide a much needed concrete view of the current state of affairs and will be valuable reference points when measuring algorithmic progress over time.

- In Chapter 8, we have designed a novel and stimulating robotics environment, which possesses one interesting property. Namely, as approximated herein by us, it can be sensibly partitioned in no less than the astronomical figure of 4.5×10^{35} abstract states per reward. In such a domain, we have then tested three algorithms reported to be proficient at handling sparse rewards, all reliant on bonus-based exploration. Our experiments revealed that these methods were hopeless in our proposed task. Indeed, after long training periods, none managed to come close to its solution. In our subsequent analysis, we reason that this underwhelming result is a consequence of their unbiased approach toward exploration. That is, to solve a sparse-reward problem such algorithms ought to evenly visit every inch of world until finding some reward. A strategy that clearly falls short in our gargantuan environment and other unassuming robotics tasks like it. We have finally estimated that with respect to present-day technology we need 10 to 20 orders of magnitude more supercomputing power before being able to solve our problem with these benchmarked techniques. Looking forward, we see as a promising research direction the introduction of subtle biases into the bonus structures of the former methods. Ideally, these would mimic those employed by people and would help to significantly reduce the exploration space in tasks involving humanoid robots.

- Chapter 9 proposes a new a method, named BESEC, founded in the successful algorithm EC. Just like its precursor, BESEC is able to handle very sparse domains; yet, it also shines in environments proven too difficult for EC. Namely, in those that deceive this latter algorithm into mastering easy-to-explore paths that take it away from desired goals and into inescapable local minima at their dead-ends. BESEC circumvents this issue by having well-designed backtracking bonuses that push it to travel back from any dead-end and to continue exploring the rest of its world. Our experiments demonstrate the effectiveness of our approach, but also its robustness to a variety of settings. We executed it while training its reachability and policy networks sequentially as well as simultaneously, and on both occasions its performance was highly satisfactory. On top of that, when pitted against a noisy TV, BESEC responded adequately by eventually paying no attention to this distractor and focusing on solving the underlying task.

Bibliography

[1] Achiam, J., Edwards, H., Amodei, D., and Abbeel, P. Variational option discovery algorithms. *CoRR*, **abs/1807.10299** (2018). arXiv:1807.10299.

[2] Andrychowicz, M., et al. Hindsight experience replay. In *NIPS* (2017).

[3] Atari VCS/2600 scoreboard. http://www.ataricompendium.com/game_library/high_scores/high_scores.html (2021). Accessed: 2021-12-30.

[4] Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. *Proceedings of the AAAI Conference on Artificial Intelligence*, **31** (2017). Available from: https://ojs.aaai.org/index.php/AAAI/article/view/10916.

[5] Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. Emergent tool use from multi-agent autocurricula. In *International Conference on Learning Representations* (2020). Available from: https://openreview.net/forum?id=SkxpxJBKwS.

[6] Beattie, C., et al. Deepmind lab. *CoRR*, **abs/1612.03801** (2016). Available from: http://arxiv.org/abs/1612.03801, arXiv:1612.03801.

[7] Bellemare, M. G., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning* (edited by D. Precup and Y. W. Teh), vol. 70 of *Proceedings of Machine Learning Research*, pp. 449–458. PMLR (2017). Available from: https://proceedings.mlr.press/v70/bellemare17a.html.

[8] Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. In *J. Artif. Intell. Res.* (2013).

[9] Bellemare, M. G., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, p. 1479–1487. Curran Associates Inc., Red Hook, NY, USA (2016). ISBN 9781510838819.

[10] Beyer, L., Vincent, D., Teboul, O., Gelly, S., Geist, M., and Pietquin, O. MULEX: disentangling exploitation from exploration in deep RL. *CoRR*, **abs/1907.00868** (2019). arXiv:1907.00868.

[11] Burda, Y., Edwards, H., Pathak, D., Storkey, A. J., Darrell, T., and Efros, A. A. Large-scale study of curiosity-driven learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net (2019). Available from: https://openreview.net/forum?id=rJNwDjAqYX.

[12] Burda, Y., Edwards, H., Storkey, A. J., and Klimov, O. Exploration by random network distillation. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net (2019). Available from: https://openreview.net/forum?id=H1lJJnR5Ym.

[13] Catacora Ocana, J. M., Capobianco, R., and Nardi, D. Exploration-intensive distractors: Two environment proposals and a benchmarking. In *AIxIA - 2021 Advances in Artificial Intelligence - XXth International Conference of the Italian Association for Artificial Intelligence, Virtual Event, December 1-3* (edited by S. Bandini, F. Gasparini, V. Mascardi, M. Palmonari, and G. Vizzari) (2022).

[14] Chevalier-Boisvert, M., Willems, L., and Pal, S. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid (2018).

[15] Clark, J. and Amodei, D. Faulty reward functions in the wild. https://openai.com/blog/faulty-reward-functions/ (2016).

[16] Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning* (edited by H. D. III and A. Singh), vol. 119 of *Proceedings of Machine Learning Research*, pp. 2048–2056. PMLR (2020). Available from: https://proceedings.mlr.press/v119/cobbe20a.html.

[17] Colas, C., Fournier, P., Sigaud, O., Chetouani, M., and Oudeyer, P.-Y. Curious: Intrinsically motivated modular multi-goal reinforcement learning. In *ICML 2019 - Thirty-sixth International Conference on Machine Learning*, pp. 1331–1340 (2019).

[18] Dann, M., Zambetta, F., and Thangarajah, J. Deriving subgoals autonomously to accelerate learning in sparse reward domains. *Proceedings of the AAAI Conference on Artificial Intelligence*, **33** (2019), 881. Available from: https://ojs.aaai.org/index.php/AAAI/article/view/3876, doi:10.1609/aaai.v33i01.3301881.

[19] Du, S. S., Kakade, S. M., Wang, R., and Yang, L. F. Is a good representation sufficient for sample efficient reinforcement learning? In *International Conference on Learning Representations* (2020). Available from: https://openreview.net/forum?id=r1genAVKPB.

[20] Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. Benchmarking deep reinforcement learning for continuous control. In *Proceedings of The 33rd International Conference on Machine Learning* (edited by M. F. Balcan and K. Q. Weinberger), vol. 48 of *Proceedings of Machine Learning Research*, pp. 1329–1338. PMLR, New York, New York, USA (2016). Available from: https://proceedings.mlr.press/v48/duan16.html.

[21] Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. First return, then explore. *Nature*, **590** (2021), 580. Available from: https://doi.org/10.1038/s41586-020-03157-9, doi:10.1038/s41586-020-03157-9.

[22] Espeholt, L., et al. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning* (edited by J. Dy and A. Krause), vol. 80 of *Proceedings of Machine Learning Research*, pp. 1407–1416. PMLR (2018). Available from: https://proceedings.mlr.press/v80/espeholt18a.html.

[23] Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations* (2019). Available from: https://openreview.net/forum?id=SJx63jRqFm.

[24] Eysenbach, B., Salakhutdinov, R. R., and Levine, S. Search on the replay buffer: Bridging planning and reinforcement learning. In *Advances in Neural Information Processing Systems 32* (edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett), pp. 15220–15231. Curran Associates, Inc. (2019).

[25] Farebrother, J., Machado, M. C., and Bowling, M. Generalization and regularization in DQN. *CoRR*, **abs/1810.00123** (2018). Available from: http://arxiv.org/abs/1810.00123, arXiv:1810.00123.

[26] Flet-Berliac, Y., Ferret, J., Pietquin, O., Preux, P., and Geist, M. Adversarially guided actor-critic. In *International Conference on Learning Representations* (2021). Available from: https://openreview.net/forum?id=_mQp5cr_iNy.

[27] Florensa, C., Degrave, J., Heess, N., Springenberg, J. T., and Riedmiller, M. A. Self-supervised learning of image embedding for continuous control. *CoRR*, **abs/1901.00943** (2019). Available from: http://arxiv.org/abs/1901.00943, arXiv:1901.00943.

[28] Florensa, C., Held, D., Geng, X., and Abbeel, P. Automatic goal generation for reinforcement learning agents. In *Proceedings of the 35th International Conference on Machine Learning* (edited by J. Dy and A. Krause), vol. 80 of *Proceedings of Machine Learning Research*, pp. 1515–1528. PMLR (2018). Available from: http://proceedings.mlr.press/v80/florensa18a.html.

[29] Florensa, C., Held, D., Wulfmeier, M., Zhang, M., and Abbeel, P. Reverse curriculum generation for reinforcement learning. In *Proceedings of the 1st Annual Conference on Robot Learning* (edited by S. Levine, V. Vanhoucke, and K. Goldberg), vol. 78 of *Proceedings of Machine Learning Research*, pp. 482–495. PMLR (2017). Available from: http://proceedings.mlr.press/v78/florensa17a.html.

[30] Fortunato, M., et al. Noisy networks for exploration. In *International Conference on Learning Representations* (2018). Available from: https://openreview.net/forum?id=rywHCPkAW.

[31] Fox, L., Choshen, L., and Loewenstein, Y. Dora the explorer: Directed outreaching reinforcement action-selection. *ArXiv*, **abs/1804.04012** (2018).

[32] French, R. M. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, **3** (1999), 128. Available from: https://www.sciencedirect.com/science/article/pii/S1364661399012942, doi:https://doi.org/10.1016/S1364-6613(99)01294-2.

[33] Ghosh, D., Gupta, A., and Levine, S. Learning actionable representations with goal conditioned policies. In *International Conference on Learning Representations* (2019). Available from: https://openreview.net/forum?id=Hye9lnCct7.

[34] Gregor, K., Rezende, D. J., and Wierstra, D. Variational intrinsic control. *CoRR*, **abs/1611.07507** (2016). arXiv:1611.07507.

[35] Grinsztajn, N., Ferret, J., Pietquin, O., Preux, P., and Geist, M. There is no turning back: A self-supervised approach for reversibility-aware reinforcement learning. In *Thirty-Fifth Conference on Neural Information Processing Systems* (2021). Available from: https://openreview.net/forum?id=3X65eaS4PtP.

[36] Gu, S., Holly, E., Lillicrap, T. P., and Levine, S. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pp. 3389–3396. IEEE (2017). Available from: https://doi.org/10.1109/ICRA.2017.7989385, doi:10.1109/ICRA.2017.7989385.

[37] Haarnoja, T., Ha, S., Zhou, A., Tan, J., Tucker, G., and Levine, S. Learning to walk via deep reinforcement learning. In *Robotics: Science and Systems* (2019). Available from: https://doi.org/10.15607/RSS.2019.XV.011.

[38] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ArXiv*, **abs/1801.01290** (2018).

[39] Haber, N., Mrowca, D., Wang, S., Fei-Fei, L., and Yamins, D. L. K. Learning to play with intrinsically-motivated, self-aware agents. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, p. 8398–8409. Curran Associates Inc., Red Hook, NY, USA (2018).

[40] Harb, J., Bacon, P.-L., Klissarov, M., and Precup, D. When waiting is not an option : Learning options with a deliberation cost. In *AAAI* (2017).

[41] Hasselt, H. v., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, p. 2094–2100. AAAI Press (2016).

[42] Hessel, M., et al. Rainbow: Combining improvements in deep reinforcement learning. In *AAAI* (edited by S. A. McIlraith and K. Q. Weinberger), pp. 3215–3222. AAAI Press (2018). Available from: `http://dblp.uni-trier.de/db/conf/aaai/aaai2018.html#HesselMHSODHPAS18`.

[43] Horgan, D., Quan, J., Budden, D., Barth-Maron, G., Hessel, M., van Hasselt, H., and Silver, D. Distributed prioritized experience replay. In *International Conference on Learning Representations* (2018). Available from: `https://openreview.net/forum?id=H1Dy---0Z`.

[44] Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, p. 1117–1125. Curran Associates Inc., Red Hook, NY, USA (2016). ISBN 9781510838819.

[45] Howard, R. A. *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA (1960).

[46] Juliani, A., Khalifa, A., Berges, V.-P., Harper, J., Teng, E., Henry, H., Crespi, A., Togelius, J., and Lange, D. Obstacle tower: A generalization challenge in vision, control, and planning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, p. 2684–2691. AAAI Press (2019). ISBN 9780999241141.

[47] Kapturowski, S., Ostrovski, G., Dabney, W., Quan, J., and Munos, R. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations* (2019). Available from: `https://openreview.net/forum?id=r1lyTjAqYX`.

[48] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, pp. 341–348. IEEE, Santorini, Greece (2016). The best paper award. Available from: `http://arxiv.org/abs/1605.02097`.

[49] Kim, Y., Nam, W., Kim, H., Kim, J.-H., and Kim, G. Curiosity-bottleneck: Exploration by distilling task-specific novelty. In *Proceedings of the 36th International Conference on Machine Learning* (edited by K. Chaudhuri and R. Salakhutdinov), vol. 97 of *Proceedings of Machine Learning Research*, pp. 3379–3388. PMLR (2019). Available from: `https://proceedings.mlr.press/v97/kim19c.html`.

[50] Konda, V. R. and Tsitsiklis, J. N. Actor-critic algorithms. In *Advances in Neural Information Processing Systems 12* (edited by S. A. Solla, T. K. Leen, and K. Müller), pp. 1008–1014. MIT Press (2000).

[51] Kuettler, H., Nardelli, N., Miller, A. H., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktaeschel, T. The NetHack Learning Environment. In *Proceedings of the Conference on Neural Information Processing Systems (NeurIPS)* (2020).

[52] Lanier, J. B., McAleer, S., and Baldi, P. Curiosity-driven multi-criteria hindsight experience replay. *CoRR*, **abs/1906.03710** (2019). Available from: `http://arxiv.org/abs/1906.03710`, `arXiv:1906.03710`.

[53] Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., and Srinivas, A. Reinforcement learning with augmented data. In *Advances in Neural Information Processing Systems* (edited by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin), vol. 33, pp. 19884–19895. Curran Associates, Inc. (2020). Available from: `https://proceedings.neurips.cc/paper/2020/file/e615c82aba461681ade82da2da38004a-Paper.pdf`.

[54] Laskin, M., Srinivas, A., and Abbeel, P. Curl: Contrastive unsupervised representations for reinforcement learning. *Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119*, (2020). ArXiv:2004.04136.

[55] Laversanne-Finot, A., Péré, A., and Oudeyer, P. Curiosity driven exploration of learned disentangled goal spaces. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, vol. 87 of *Proceedings of Machine Learning Research*, pp. 487–504. PMLR (2018). Available from: `http://proceedings.mlr.press/v87/laversanne-finot18a.html`.

[56] Lee, K., Lee, K., Shin, J., and Lee, H. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations* (2020). Available from: `https://openreview.net/forum?id=HJgcvJBFvB`.

[57] Lee, L., Eysenbach, B., Parisotto, E., Xing, E. P., Levine, S., and Salakhutdinov, R. Efficient exploration via state marginal matching. *CoRR*, **abs/1906.05274** (2019). Available from: `http://arxiv.org/abs/1906.05274`, `arXiv:1906.05274`.

[58] Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, **17** (2015), 39:1.

[59] Levy, A., Platt, R., and Saenko, K. Hierarchical reinforcement learning with hindsight. In *International Conference on Learning Representations* (2019). Available from: https://openreview.net/forum?id=ryzECoAcY7.

[60] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. In *ICLR* (edited by Y. Bengio and Y. LeCun) (2016). Available from: http://dblp.uni-trier.de/db/conf/iclr/iclr2016.html#LillicrapHPHETS15.

[61] Liu, H., Trott, A., Socher, R., and Xiong, C. Competitive experience replay. In *International Conference on Learning Representations* (2019).

[62] Lyu, D., Yang, F., Liu, B., and Gustafson, S. Sdrl: Interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. *Proceedings of the AAAI Conference on Artificial Intelligence*, **33** (2019), 2970. Available from: https://ojs.aaai.org/index.php/AAAI/article/view/4153, doi:10.1609/aaai.v33i01.33012970.

[63] Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *J. Artif. Int. Res.*, **61** (2018), 523–562.

[64] Martin, J., Sasikumar, S. N., Everitt, T., and Hutter, M. Count-based exploration in feature space for reinforcement learning. In *IJCAI* (2017).

[65] Mnih, V., et al. Human-level control through deep reinforcement learning. *Nature*, **518** (2015), 529.

[66] Mohamed, S. and Rezende, D. J. Variational information maximisation for intrinsically motivated reinforcement learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, p. 2125–2133. MIT Press, Cambridge, MA, USA (2015).

[67] Nachum, O., Gu, S., Lee, H., and Levine, S. Data-efficient hierarchical reinforcement learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, p. 3307–3317. Curran Associates Inc., Red Hook, NY, USA (2018).

[68] Nair, A., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. Visual reinforcement learning with imagined goals. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, p. 9209–9220. Curran Associates Inc., Red Hook, NY, USA (2018).

[69] O'Donoghue, B., Osband, I., Munos, R., and Mnih, V. The uncertainty bellman equation and exploration. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018* (edited by J. G. Dy and A. Krause), vol. 80 of *Proceedings of Machine Learning Research*, pp. 3836–3845. PMLR (2018). Available from: http://proceedings.mlr.press/v80/o-donoghue18a.html.

[70] Oh, J., Guo, Y., Singh, S., and Lee, H. Self-imitation learning. In *Proceedings of the 35th International Conference on Machine Learning* (edited by J. Dy and A. Krause), vol. 80 of *Proceedings of Machine Learning Research*, pp. 3878–3887. PMLR (2018). Available from: http://proceedings.mlr.press/v80/oh18b.html.

[71] OpenAI, et al. Dota 2 with large scale deep reinforcement learning. (2019). Available from: https://arxiv.org/abs/1912.06680, arXiv:1912.06680.

[72] Osband, I., Blundell, C., Pritzel, A., and Roy, B. V. Deep exploration via bootstrapped dqn. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, p. 4033–4041. Curran Associates Inc., Red Hook, NY, USA (2016). ISBN 9781510838819.

[73] Ostrovski, G., Bellemare, M. G., van den Oord, A., and Munos, R. Count-based exploration with neural density models. In *ICML* (2017).

[74] Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, (2017), 488.

[75] Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S. Skew-fit: State-covering self-supervised reinforcement learning. *CoRR*, **abs/1903.03698** (2019). arXiv:1903.03698.

[76] Puigdomenech Badia, A., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, vol. 119 of *Proceedings of Machine Learning Research*, pp. 507–517. PMLR (2020). Available from: http://proceedings.mlr.press/v119/badia20a.html.

[77] Puigdomenech Badia, A., et al. Never give up: Learning directed exploration strategies. In *International Conference on Learning Representations* (2020). Available from: https://openreview.net/forum?id=Sye57xStvB.

[78] Raileanu, R., Goldstein, M., Yarats, D., Kostrikov, I., and Fergus, R. Automatic data augmentation for generalization in deep reinforcement learning. *CoRR*, **abs/2006.12862** (2020). Available from: https://arxiv.org/abs/2006.12862, arXiv:2006.12862.

[79] Raileanu, R. and Rocktaeschel, T. Ride: Rewarding impact-driven exploration for procedurally-generated environments. In *International Conference on Learning Representations* (2020). Available from: https://openreview.net/forum?id=rkg-TJBFPB.

[80] Rauber, P., Ummadisingu, A., Mutz, F., and Schmidhuber, J. Hindsight policy gradients. In *International Conference on Learning Representations* (2019).

[81] Ren, Z., Dong, K., Zhou, Y., Liu, Q., and Peng, J. Exploration via hindsight goal generation. In *Advances in Neural Information Processing Systems 32* (edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett), pp. 13464–13474. Curran Associates, Inc. (2019).

[82] Riemer, M., Liu, M., and Tesauro, G. Learning abstract options. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, p. 10445–10455. Curran Associates Inc., Red Hook, NY, USA (2018).

[83] Roderick, M., Grimm, C., and Tellex, S. Deep abstract q-networks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '18, p. 131–138. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2018).

[84] Savinov, N., Raichuk, A., Vincent, D., Marinier, R., Pollefeys, M., Lillicrap, T., and Gelly, S. Episodic curiosity through reachability. In *International Conference on Learning Representations* (2019). Available from: `https://openreview.net/forum?id=SkeK3s0qKQ`.

[85] Schmidhuber, J. Curious model-building control systems. In *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, pp. 1458–1463 vol.2 (1991).

[86] Schrittwieser, J., et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature*, **588** (2020), 604. `doi:10.1038/s41586-020-03051-4`.

[87] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning* (edited by F. Bach and D. Blei), vol. 37 of *Proceedings of Machine Learning Research*, pp. 1889–1897. PMLR, Lille, France (2015). Available from: `https://proceedings.mlr.press/v37/schulman15.html`.

[88] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, **abs/1707.06347** (2017). Available from: `http://arxiv.org/abs/1707.06347`, `arXiv:1707.06347`.

[89] Schwarzer, M., Anand, A., Goel, R., Hjelm, R. D., Courville, A., and Bachman, P. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations* (2021). Available from: `https://openreview.net/forum?id=uCQfPZwRaUu`.

[90] Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. Planning to explore via self-supervised world models. In *ICML* (2020).

[91] Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. Dynamics-aware unsupervised discovery of skills. *CoRR*, **abs/1907.01657** (2019). `arXiv:1907.01657`.

[92] Shyam, P., Jaśkowski, W., and Gomez, F. Model-based active exploration. In *Proceedings of the 36th International Conference on Machine Learning* (edited by K. Chaudhuri and R. Salakhutdinov), vol. 97 of *Proceedings of Machine Learning Research*, pp. 5779–5788. PMLR (2019). Available from: https://proceedings.mlr.press/v97/shyam19a.html.

[93] Silver, D., et al. Mastering the game of go without human knowledge. *Nature*, **550** (2017), 354.

[94] Simmons-Edler, R., Eisner, B., Mitchell, E., Seung, H. S., and Lee, D. D. Qxplore: Q-learning exploration by maximizing temporal difference error. *CoRR*, **abs/1906.08189** (2019). arXiv:1906.08189.

[95] Smith, M., van Hoof, H., and Pineau, J. An inference-based policy gradient method for learning options. In *Proceedings of the 35th International Conference on Machine Learning* (edited by J. Dy and A. Krause), vol. 80 of *Proceedings of Machine Learning Research*, pp. 4703–4712. PMLR, Stockholmsmässan, Stockholm Sweden (2018).

[96] Song, Y., Wang, J., Lukasiewicz, T., Xu, Z., Zhang, S., Wojcicki, A., and Xu, M. Mega-reward: Achieving human-level play without extrinsic rewards. *Proceedings of the AAAI Conference on Artificial Intelligence*, **34** (2020), 5826. Available from: https://ojs.aaai.org/index.php/AAAI/article/view/6040, doi:10.1609/aaai.v34i04.6040.

[97] Stadie, B. C., Levine, S., and Abbeel, P. Incentivizing exploration in reinforcement learning with deep predictive models. *CoRR*, **abs/1507.00814** (2015). Available from: http://arxiv.org/abs/1507.00814, arXiv:1507.00814.

[98] Stanton, C. and Clune, J. Deep curiosity search: Intra-life exploration improves performance on challenging deep reinforcement learning problems. *CoRR*, **abs/1806.00553** (2018). Available from: http://arxiv.org/abs/1806.00553, arXiv:1806.00553.

[99] Sukhbaatar, S., Denton, E., Szlam, A., and Fergus, R. Learning goal embeddings via self-play for hierarchical reinforcement learning. *CoRR*, **abs/1811.09083** (2018). arXiv:1811.09083.

[100] Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R. Intrinsic motivation and automatic curricula via asymmetric self-play. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net (2018). Available from: https://openreview.net/forum?id=SkT5Yg-RZ.

[101] Sutton, R. S. Learning to predict by the methods of temporal differences. *Mach. Learn.*, **3** (1988), 9–44.

[102] Sutton, R. S. and Barto, A. G. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edn. (1998).

[103] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, p. 1057–1063. MIT Press, Cambridge, MA, USA (1999).

[104] Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, **112** (1999), 181–211.

[105] Taiga, A. A., Fedus, W., Machado, M. C., Courville, A., and Bellemare, M. G. On bonus based exploration methods in the arcade learning environment. In *International Conference on Learning Representations* (2020). Available from: https://openreview.net/forum?id=BJewlyStDr.

[106] Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., Turck, F. D., and Abbeel, P. #exploration: A study of count-based exploration for deep reinforcement learning. In *NIPS* (2016).

[107] Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033 (2012). doi:10.1109/IROS.2012.6386109.

[108] van Hasselt, H. P., Hessel, M., and Aslanides, J. When to use parametric models in reinforcement learning? In *Advances in Neural Information Processing Systems* (edited by H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett), vol. 32. Curran Associates, Inc. (2019). Available from: https://proceedings.neurips.cc/paper/2019/file/1b742ae215adf18b75449c6e272fd92d-Paper.pdf.

[109] Veeriah, V., Oh, J., and Singh, S. Many-goals reinforcement learning. *CoRR*, **abs/1806.09605** (2018). Available from: http://arxiv.org/abs/1806.09605, arXiv:1806.09605.

[110] Venkattaramanujam, S., Crawford, E. W., Doan, T., and Precup, D. Self-supervised learning of distance functions for goal-conditioned reinforcement learning. *ArXiv*, **abs/1907.02998** (2019).

[111] Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, p. 3540–3549. JMLR.org (2017).

[112] Vinyals, O., et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, **575** (2019), 350. doi:10.1038/s41586-019-1724-z.

[113] Wang, J. X., et al. Alchemy: A structured task distribution for meta-reinforcement learning. *CoRR*, **abs/2102.02926** (2021). Available from: https://arxiv.org/abs/2102.02926, arXiv:2102.02926.

[114] Wen, Z., Precup, D., Ibrahimi, M., Barreto, A., Van Roy, B., and Singh, S. On efficiency in hierarchical reinforcement learning. In *Advances in Neural Information Processing Systems* (edited by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin), vol. 33, pp. 6708–6718. Curran Associates, Inc. (2020). Available from: https://proceedings.neurips.cc/paper/2020/file/4a5cfa9281924139db466a8a19291aff-Paper.pdf.

[115] Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, **8** (1992), 229–256.

[116] Wu, Y., Tucker, G., and Nachum, O. The laplacian in RL: learning representations with efficient approximations. *CoRR*, **abs/1810.04586** (2018). arXiv:1810.04586.

[117] Wulfmeier, M., et al. Compositional Transfer in Hierarchical Reinforcement Learning. In *Proceedings of Robotics: Science and Systems*. Corvalis, Oregon, USA (2020). doi:10.15607/RSS.2020.XVI.054.

[118] Yang, Y., Caluwaerts, K., Iscen, A., Zhang, T., Tan, J., and Sindhwani, V. Data efficient reinforcement learning for legged robots. In *Proceedings of the Conference on Robot Learning* (edited by L. P. Kaelbling, D. Kragic, and K. Sugiura), vol. 100 of *Proceedings of Machine Learning Research*, pp. 1–10. PMLR (2020). Available from: https://proceedings.mlr.press/v100/yang20a.html.

[119] Zambaldi, V., et al. Deep reinforcement learning with relational inductive biases. In *International Conference on Learning Representations* (2019). Available from: https://openreview.net/forum?id=HkxaFoC9KQ.

[120] Zhang, S. and Whiteson, S. DAC: the double actor-critic architecture for learning options. *CoRR*, **abs/1904.12691** (2019). arXiv:1904.12691.

[121] Łukasz Kaiser, et al. Model based reinforcement learning for atari. In *International Conference on Learning Representations* (2020). Available from: https://openreview.net/forum?id=S1xCPJHtDB.