Skill Discovery for Exploration and Planning

By

Akhil Bagaria

B.S., Harvey Mudd College, 2016

Sc.M., Brown University, 2022

Thesis

Submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in the Department of Computer Science at Brown University

PROVIDENCE, RHODE ISLAND

May 2025

© Copyright 2024 Akhil Bagaria

This dissertation by Akhil Bagaria is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date ______________ ____

George Konidaris, Advisor

Recommended to the Graduate Council

Date ______________ ____

Michael Littman, Reader

Date ______________ ____

Richard Sutton, Reader

Approved by the Graduate Council

Date ______________ ____

Thomas A. Lewis, Dean of the Graduate School

Curriculum Vitae

Akhil Bagaria grew up in Kolkata, India. He studied Engineering, with an emphasis on Electrical Engineering, at Harvey Mudd College, in Claremont California. He graduated Harvey Mudd with Honors in 2016 with an award for demonstrating outstanding engineering judgment. After graduating, he worked at Apple as a Sensor Algorithms Engineer for 2 years in Cupertino, California. At Apple, he developed multi-touch algorithms for trackpads in Macbook laptops, and text editing interactions for iPad tablets. Then, he attended Brown University in Providence, Rhode Island, where he got a Masters degree in Computer Science en route to his Ph.D. During his Ph.D, he did an internship at DeepMind, London. He is currently employed as a Senior Applied Scientist at Amazon.

During his Ph.D, Akhil published the following conference papers: *Option Discovery using Deep Skill Chaining* (Bagaria and Konidaris, 2020a), *Robustly Learning Composable Options in Deep Reinforcement Learning*, (Bagaria et al., 2021a), *Skill Discovery for Exploration and Planning* (Bagaria et al., 2021b), *Optimistic Initialization for Exploration in Continuous Control* (Lobel et al., 2022) *Flipping Coins to Estimate Pseudocounts for Exploration in Reinforcement Learning* (Lobel et al., 2023), *Scaling Goal-based Exploration via Pruning Proto-goals* (Bagaria and Schaul, 2023), *Effectively Learning Initiation Sets in Hierarchical Reinforcement Learning* (Bagaria et al., 2023), and *Discovering Options that Minimize Average Planning Time* (Ivanov et al., 2024).

For my family

Acknowledgments

Before my Ph.D, I never viewed science as a creative endeavor. This shift in perspective is largely thanks to my advisor, George Konidaris. Thank you, George, for this gift. Thank you for asking the big questions and encouraging me to do the same. Thank you for taking a chance on me all those years ago, and for bringing me along on this impossibly hard, yet fun journey. I too have caught the bug now, and hope to approach my career with your unique blend of ambition, humility, and kindness.

Thank you, Michael Littman and Stefanie Tellex, for creating a supportive and collaborative environment for AI research at Brown, and for letting me be a part of it. Your advice and feedback have shaped much of my research and this thesis.

Thank you to my external committee member, Rich Sutton, for being such a profound inspiration. I read your textbook on reinforcement learning on a flight as a college student, and have been captivated by the beauty and simplicity of your ideas ever since. Every day I get to think about reinforcement learning is a joyful day, and I have you (and Andy) to thank for it. The days surrounding my defense, spent with you in Providence, stand as some of the most intellectually stimulating moments of my life. Thank you for making me feel that my ideas matter.

Thank you to Tom Schaul, my mentor during my DeepMind internship. When I arrived in London, my thoughts and ideas were all over the place. Through our numerous coffee chats, you helped me realize the importance of exploration, and how temporal abstraction fit into that larger picture. Each week during my internship with you, I learned immensely and had a ton of fun doing so. I am also grateful to the other DeepMinders—John Quan, Georg Ostrovski, David Silver, Will Dabney, Vivek Veeriah, Adrià Badia, Jake Bruce, Hado van Hasselt, and Chentian Jiang—for making my five months at DeepMind so intellectually rewarding.

Thank you to my professors at that special little place, Harvey Mudd: Profs Clark, Harris, Spencer, Dodds, Jakes, Duron, Olson, Ben, Boerkel, Lape, Yong, Sumi, Dadabhoy, and so many others for sharing your passion for science, engineering, and teaching. Thank you also for going out of your way to help me even when you didn't have to——whether sitting with me late into the night until I actually understood the material, or being kind and understanding when I was overwhelmed. To my Mudder friends: Ben, Appi, Mo, Fabs, Kelly, Kunal, Paul, Obosa, Alistair, Sonya, Mai, Trang, Shannon, and so many more—without you I would not have survived Mudd, and this thesis would not have been possible.

A big thank you to my fellow lab mates for being great friends and wonderful collaborators. Thank you, Sam, for your friendship, optimism, and couch. The time we spent working on CFN serves as a model for collaborations I aspire to have throughout my research career—building on each other's ideas, pair programming, laughing, and playing too much Overcooked. Thank you, Rafa and Saket, for being the reasons I loved my life in Providence—seeing you daily made rough times smoother and good times more meaningful. Thank you also to my lab seniors: Dave Abel, Nakul Gopalan, Kavosh Asadi, Omer Gottesman, and Cam Allen for your advice and encouragement.

Shifting to my personal life. First, a big thanks to Tessa, for so many years of friendship and support. You have always pushed me to take risks and bet on myself—thank you for encouraging me to pursue my Ph.D. Thank you for introducing me to Smokey, and thank you, Smokey, for your overwhelming love and cuteness. I would also like to thank the Bartons—Liz, Rich, and Joel—for letting me stay in your beautiful Montana home during the pandemic, and making me feel like family. You were my home away from home.

Thank you to my friends in Providence: Esen, Sena, Lefteris, Abaho, Alessio, Gabby, Geetika, Naman, Tallie, Ji Won, and so many others, for way too many hours at the GCB, and for filling Providence with wonderful, lifelong memories.

Finally, I would like to thank my family. Your support never wavered, regardless of how strange my goals might have seemed—whether moving halfway across the world at seventeen or leaving a well-paying job to pursue a Ph.D in an unfamiliar field for six years. This thesis is dedicated to your unending love and support.

Abstract of *Skill Discovery for Exploration and Planning*, by Akhil Bagaria, Ph.D., Brown University, May 2025.

General-purpose agents must sense the world using high-dimensional sensors such as cameras and affect the world using low-level actions such as motor torques. Such a rich input and output space allows the same agent to solve many tasks, but complicates the efficient search for solutions. To solve any specific problem efficiently, the agent must form *abstractions*—only retain the information necessary to make good decisions in the current task. Sequential decision-making problems permit two types of abstraction: state and action abstractions; this thesis studies how artificial agents may autonomously acquire both types of abstraction via interaction with an environment.

Hierarchical reinforcement learning (HRL) is our starting point for action abstraction because it augments the agent's action set with temporally extended actions called *skills*. When a human programmer carefully designs temporally extended skills for an AI agent, it can rapidly solve many challenging tasks. But, how can an agent automatically discover skills that are useful for obtaining mastery over the environment? We present algorithms that discover skills that are temporally composable, meaning that they can be reliably executed one after the other to solve long-horizon problems.

Action abstraction, although necessary, is not sufficient for effective decision making; it must be accompanied by state abstractions. We provide algorithms that can learn state abstractions that complement the skills discovered by the agent so that it may create a whole new decision problem that is easier to solve than the original one it was faced with. Solutions in the abstract decision problem correspond to good solutions in the original problem, providing a way for artificial agents to rapidly make high-quality decisions. These algorithms bridge the gap between classical AI techniques that specialize in symbolic search, and deep reinforcement learning techniques that effectively deal with high-dimensional and continuous sensorimotor spaces.

Finally, we tackle the problem of exploration in vast environments: when the agent interacts with a large environment, what data should it collect so that it can maximize its learning progress? Here, we present algorithms for discovering those goals that are controllable, reachable, novel, and task-relevant as a way of guiding the curiosity of the agent. Taken together, this thesis presents algorithmic ingredients for building agents that explore with the intention of building plannable models of the world that are abstract in both state and time.

Contents

Acknowledgments **vi**

1 INTRODUCTION **1**

1.1 Reinforcement Learning and Abstractions 3

1.2 Exploration in Reinforcement Learning 5

1.3 Contributions 6

2 Background and Related Work **8**

2.1 Reinforcement Learning 8

2.1.1 Markov Decision Processes 9

2.2 Planning 11

2.3 Learning from Interaction 13

2.3.1 Model-free RL 13

2.3.2 Model-based RL and Exploration 15

2.4 General Value Functions 18

2.5 The Options Framework 19

2.5.1 Option Discovery 20

2.5.2 Connection between Options and GVFs 22

2.6 Goal-conditioned RL 23

2.6.1 Goal-conditioned RL is Closely Linked to Options 24

2.6.2 Hindsight Experience Replay 24

2.7 Related Work on Option Discovery 25
2.7.1 Spectral Methods for Option Discovery 25
2.7.2 Empowerment based Skill Discovery 27
2.7.3 Option Discovery from Extrinsic Rewards 30
2.7.4 Discovery using Meta-Learning 33
2.7.5 Option Discovery with Formal Guarantees 35
2.8 Summary and Conclusions 38

3 Option Discovery using Deep Skill Chaining 40
3.1 Method 41
3.1.1 Intra-Option Policy 42
3.1.2 Policy Over Options 42
3.1.3 Initiation Set Classifier 44
3.1.4 Generalizing to Skill Trees 45
3.1.5 Optimality of Discovered Solutions 45
3.2 Experiments 46
3.2.1 Comparative Analyses 48
3.2.2 Interpreting Learned Skills 50
3.3 Discussion and Conclusion 50

4 Robustly Learning Composable Options in Deep Reinforcement Learning 53
4.1 Robustly Learning Composable Options 55
4.1.1 Dual Initiation Classifiers to Avoid Shrinkage 56
4.1.2 Robust Subgoals via Goal State Selection 56
4.1.3 Learning Robust Option Policies 58
4.2 Experiments 59
4.2.1 Implementation Details 60
4.2.2 Evaluating Robustness 60

4.2.3 Evaluating Hierarchical Optimality 62
4.2.4 Learning Curves 63
4.3 Conclusion 65

5 Effectively Learning Initiation Sets in Hierarchical Reinforcement Learning 66
5.1 Effectively Learning Initiation Sets 68
5.1.1 Initiation Value Function (IVF) 69
5.1.2 Combining Classification and the Initiation Value Function 71
5.1.3 Overcoming Pessimistic Bias 72
5.1.4 Pessimistic Bias: Connections to Non-Hierarchical MDPs 73
5.2 Experiments 74
5.2.1 Measuring the Quality of Initiation Sets 75
5.2.2 Robot Manipulation: Identifying Promising Grasps 78
5.2.3 Improving Option Discovery 79
5.3 Conclusion 82

6 Skill Discovery for Exploration and Planning using Deep Skill Graphs 83
6.1 Deep Skill Graphs 84
6.1.1 Definitions 85
6.1.2 Overview of the Graph Construction Algorithm 86
6.1.3 Graph Expansion 87
6.1.4 Graph Consolidation 89
6.1.5 Querying the Skill Graph at Test Time 91
6.2 Experiments 91
6.2.1 Implementation Details 91
6.2.2 Qualitative Evaluation 92
6.2.3 Quantitative Evaluation 93
6.3 Results 95

6.4 Conclusion 97

7 Intrinsically Motivated Discovery of Temporally Abstract World Model Graphs 98

7.1 Background and Related Work 100

7.2 Intrinsically Motivated Deep Skill Graphs 104

7.2.1 Semantics of the Skill Graph 104

7.2.2 Graph Construction Algorithm 105

7.3 Experiments 114

7.3.1 Domain Descriptions 114

7.3.2 Qualitative Evaluation 116

7.3.3 Quantitative Evaluation 118

7.4 Discussion and Conclusion 121

8 Scaling Goal-based Exploration via Pruning Proto-goals 123

8.1 Background and Related Work 125

8.2 Goals and Proto-goals 126

8.2.1 Example Proto-goal Spaces 127

8.2.2 Representation 128

8.2.3 Goal Recombinations 128

8.3 Proto-goal Evaluator 129

8.3.1 Plausibility Pruning 129

8.3.2 Desirability Weighting 131

8.4 Integrated RL Agent 132

8.4.1 Which Goal to Pursue in the Current State 133

8.4.2 Which Goals to Learn about in Hindsight 134

8.4.3 Mastery-based Goal Recombination 134

8.4.4 Distributed RL Agent Implementation 135

8.5 Experiments 135

8.5.1 Tabular Experiment: Exploration in TAXI 136
8.5.2 Verifying the Controllability Measure 137
8.5.3 Natural Language Proto-goals: MINIHACK 139
8.5.4 Doubly Combinatorial Puzzles: BABA IS YOU 141
8.6 Conclusion and Future Work 142

9 Going Beyond State-Reaching: Learning Abstractions for Intrinsically Motivated Skill Discovery 144
9.1 Background and Related Work 146
9.1.1 Learning Subgoal Achievement Classifiers 146
9.1.2 Options of feature attainment 148
9.2 Method: Abstract Subgoal Discovery 148
9.2.1 Generating Exploratory Behaviors 150
9.2.2 Identifying Interesting Events 150
9.2.3 Feature Selection Algorithm 151
9.2.4 Classification 153
9.2.5 Policy Learning 154
9.3 Experiments 154
9.3.1 Qualitative Analysis 155
9.4 Conclusion 156

10 Discussion and Conclusion 158
10.1 Limitations and Future Work 159
10.2 Closing Remarks 161

Parts of this thesis were conducted in collaboration with others and have been previously published elsewhere.

- Chapter 3 is based on work led by myself, conducted with George Konidaris, and published in the International Conference on Learning Representations (ICLR), 2020.

- Chapter 4 is based on work led by myself, conducted with Jason Senthil, Matthew Slivinski, and George Konidaris, and published in the International Joint Conferences on Artificial Intelligence (IJCAI), 2021.
- Chapter 5 is based on work led by myself, conducted with Ben Abbatematteo, Omer Gottesman, Matt Corsaro, Sreehari Rammohan, and George Konidaris, and published in the Advances in Neural Information Processing Systems (NeurIPS), 2024.
- Chapter 6 is based on work led by myself, conducted with Jason Senthil and George Konidaris, and published in the International Conference on Machine Learning (ICML), 2021.
- Chapter 7 is based on work led by myself, conducted with Anita De Mello Koch, Rafael Rodriguez, Sam Lobel, and George Konidaris, and is currently under submission at the Reinforcement Learning Conference (RLC), 2025.
- Chapter 8 is based on work led by myself, conducted with Tom Schaul, and published in the International Joint Conferences on Artificial Intelligence (IJCAI), 2024.
- Chapter 9 is based on work led by myself, conducted with Anita De Mello Koch, and George Konidaris, and has been accepted for presentation at the Multi-Disciplinary Conference on Reinforcement Learning and Decision-Making (RLDM), 2025.

List of Figures

3.1 DSC learning curves 47
3.2 DSC learned initiation sets 48
3.3 DSC solution trajectories 51

4.1 The non-stationary subgoal problem 55
4.2 Test environments for Robust DSC 57
4.3 Evaluating robustness of skill composition 61
4.4 Subgoal sampling in Robust DSC 62
4.5 Robust initiation classifiers in Ant Reacher 63
4.6 Learning curves for Robust DSC 64
4.7 Robust DSC solution trajectory 65

5.1 Challenges of learning initiation sets online 69
5.2 Pessimistic bias of initiation set learning 73
5.3 Domains to test initiation learning 75
5.4 Measuring the quality of learned initiation sets 77
5.5 Learning curves for grasping tasks 79
5.6 Promising grasp poses learned by IVF 80
5.7 Learning curves with improved initiation sets 81

6.1 Overview of DSG algorithm 84
6.2 Overview of graph consolidation 89

6.3 Demonstration of exploration with DSG 92
6.4 Incremental expansion of the skill graph 93
6.5 DSG learning curves 94
6.6 Discovered skill graph in large Ant Maze 96

7.1 IM-DSG Illustration 99
7.2 IM-DSG Illustration of Abstract MDP 111
7.3 Domains used to test the IM-DSG agent 115
7.4 Skill graph discovered by IM-DSG 117
7.5 Policy execution in Sokoban 118
7.6 IM-DSG learning curves 119
7.7 IM-DSG Ablations and Multi-task Performance 120

8.1 Illustration of Proto-goal RL 124
8.2 Illustration of Proto-goal space in NetHack 128
8.3 Goal space refinement in SparseTaxi 135
8.4 Proto-goal learning curves in SparseTaxi 136
8.5 Proto-goal learning curves in MiniHack 137
8.6 Evaluation of Controllability GVF 138
8.7 Proto-goals learning curves in Baba Is You 139

9.1 Illustration of abstract subgoal classifiers 149
9.2 Subgoal classifier feature selection 151
9.3 Goal achievement classifiers in MiniGrid 155
9.4 Factored Options Learning Curves 156

CHAPTER 1

INTRODUCTION

Abstraction is key to decision making. Humans and other animals form abstractions—they perceive and act in a complex sensorimotor space, but they plan and reason using representations that discard vast amounts of irrelevant information to focus only on what is task-relevant (Tenenbaum et al., 2011). This ability to abstract is what allows us to effortlessly make countless decisions in a complex, high-dimensional world (Ho et al., 2021; Gershman, 2017). When AI agents are given a concise description of a specific problem, they are able to outperform humans in several high-profile domains, such as Chess and Go (Silver et al., 2017b). However, can we create general-purpose agents that acquire powerful, problem-specific abstractions themselves?

The necessity for abstractions stems from the fact that general-purpose agents must perceive the world via high-dimensional sensors such as cameras and IMUs and affect the world via low-level actions like motor voltages and torques. The vastness of this input and output space is not a luxury or a design error—it is what allows a single agent to solve a large variety of tasks (Konidaris, 2019). For example, if the same agent has to navigate the world, play chess, write a sonnet, program a computer, change a diaper, solve equations, cook a tasty meal, and so on (Heinlein, 1973), its sensorimotor space must be large enough to represent all of those tasks. Although the expansion of the agent's sensorimotor space in this way affords generality, it does not permit efficient solutions. To solve problems efficiently, we humans frame different tasks succinctly in terms of the relevant variables. For example, when playing chess, we only need to pay attention to the state of the game but may ignore information such as the material of the chess pieces, the angle from which the chess board is viewed, or even the weather outside. Imagine for a moment if that were not the case—if in addition to predicting our opponent's moves, we also had to predict the weather in Boston, surely we would be unable to play good moves. But once we are able to distill all the information needed to play chess, we can use simple heuristic search (Knuth and Moore, 1975) to play at an average human level or employ more sophisticated neural networks to play at a super-human level (Silver et al., 2017b). The challenge is therefore to automatically determine the abstractions needed, and then deploying simple solution methods that best complement those abstractions.

Broadly, decision-making problems permit two types of abstractions: state and action abstractions. *Action abstraction* generates discrete units of behavior, called *skills*, from the set of single timestep actions (or *primitive actions*) initially available to an agent. For example, a human working in an office saves a sequence of joint-level torques as a temporally extended abstract action, or skill, called "opening a door." After having done so, they can treat that skill as an atomic unit and execute it without having to specify *how* it operates. *State abstraction* retains only the relevant part of the agent's observation stream, while discarding details that are irrelevant for its current task. An agent capable of constructing both types of abstractions can represent each problem of interest as a simplified decision problem, which permits a quick, satisficing solution. This thesis will begin by focusing on how agents may acquire their own action abstractions, while progressing to methods that address the simultaneous acquisition of both state and action abstractions.

1.1 Reinforcement Learning and Abstractions

The endeavor of creating general-purpose agents that can solve a vast variety of problems in large, complex environments is elegantly captured by the reinforcement learning (RL) problem (Sutton and Barto, 2018). RL agents sense a part of the environment and must act in it at every single timestep; actions are chosen with the aim of maximizing reward, which is a real-valued number that expresses the goals and purposes of that agent (Sutton, 2004; Littman, 2017). The generality of the RL problem definition perfectly complements our desire to create general-purpose agents: although agents are severely bounded in memory and computation (compared to the vastness of the environment) (Simon, 1991; Russell and Subramanian, 1994; Sutton et al., 2022), they must incrementally improve the quality of their actions, no matter the specific characteristics of the environment or their specific goals and purposes (Silver et al., 2021).

The generality of the RL problem definition, while necessary, poses a challenge for our solution methods. In particular, our agents must confront the challenges of sensing and acting in high-dimensional spaces, while dealing with potentially large time delays between chosen actions and received rewards. Recent deep reinforcement learning algorithms (DRL) (Mnih et al., 2015) have made significant progress on some of these problems, but we are yet far from creating RL agents that can deal with the vastness and generality of the real world. A crucial ingredient missing from existing solution methods is their inability to acquire and leverage state and action abstractions.

How does the concept of learning abstractions fit into the RL framework? Hierarchical reinforcement learning (HRL) (Barto and Mahadevan, 2003) is a natural starting point for action abstraction because it augments the agent's action set with temporally extended actions called skills, which are mathematically modeled as *options* (Sutton et al., 1999). An option encodes a behavior via a policy, a way of initiating and a way of stopping. Options, unlike primitive actions, last for variable number of timesteps and can invoke the execution of other options. An agent with temporally extended options can escape the "curse of horizon", which is the problem that the quality of agent's predictions (and decisions) degrade rapidly the further ahead in the future it tries to look (Sutton et al., 1999, 2011). It has been clear for a long time that options can be immensely useful, but the core question is: where do good options come from? Can an agent discover them autonomously while interacting with the environment? This is known as the *skill discovery* problem (or equivalently, the option discovery problem), and it is the first pillar of this thesis.

Action abstraction, although necessary, is not sufficient for effective decision making; it must be complemented by state abstractions. Function approximation is a good starting point for state abstraction (indeed, this is the lesson from much of deep learning (Goodfellow et al., 2016)), but it is not enough. One reason for this is that the features learned by function approximators are not automatically amenable to *planning*, which is key to sample-efficient learning and high-quality decision-making (Tenenbaum, 2018). So, how can we learn a state abstraction that complements the options discovered by the agent while unlocking the capabilities of long-horizon planning (Konidaris et al., 2018)? This is the second pillar of this thesis.

Several other works have identified the importance of state and action abstractions (for e.g, Allen (2023); Abel (2020); Machado (2019)), but few have addressed them simultaneously. This approach misses out on a key benefit of abstraction: when learned jointly, the agent can create a whole new decision problem, but one that is easier to solve than the original one that it was faced with. If this new abstract decision problem is well "grounded" (meaning that good behaviors in the abstract problem correspond to good behaviors in the real environment), the agent can use methods from search/planning to make high quality decisions with relative ease. This drive to interleave the acquisition of both types of abstraction in a single algorithm is the third pillar of this thesis.

1.2 Exploration in Reinforcement Learning

Most machine learning paradigms are concerned with how to learn patterns in a previously acquired dataset (Bishop and Nasrabadi, 2006). However, reinforcement learning agents must additionally solve the problem of *exploration*: what data should be collected in the first place? Of course, this problem is ubiquitous in the natural world; for example, human babies have an extended period of *play*, where they seem to perform experiments that tease out how the world around them works (Gopnik et al., 1999; Gopnik, 2020). This exploration of the world is crucial: without a drive to seek out new, interesting data, an agent has no hope of understanding how the world works, let alone have a policy that achieves complex goals and purposes.

Within RL, the exploration question has often either been viewed as an afterthought (most commonly, methods follow a noisy version of their current approximation of the optimal policy) or through the lens of novelty maximization (Schmidhuber, 2010b; Chentanez et al., 2005; Brafman and Tennenholtz, 2002), wherein the reward function is augmented with a bonus (Sutton, 1990; Taïga et al., 2019) that encourages visiting new states. However, the number of possible states in the world is so large that an agent trying to visit them all is unlikely to discover anything interesting. Through this thesis, I argue that the problem of exploration is better seen as a drive to discover models of the world that are abstract in both state and time. This approach has the additional benefit that it frees the agent from having to learn an accurate one-step model of the world, which is difficult due to model prediction errors compounding over time (Janner et al., 2019; Talvitie, 2017). In particular, by discovering options and the state-abstractions that complement those options, an agent performs exploration in a way that builds relevant knowledge about the world (Sutton et al., 2011; Schaul and Ring, 2013), which in turn allows it to make better decisions. This is the fourth and final pillar of this thesis.

1.3 Contributions

This thesis makes three contributions. The first is on the question of discovering options that can be sequentially composed with high reliability. The second contribution is to build graph-based models of the world that achieve state and action abstraction. The third contribution is to forego coverage, and guide option discovery in promising subspaces of the state-space.

1. **Discovering sequentially composable skills.** Sequential composition of skills, i.e, the ability to execute one skill after another with high reliability, is central to learning skills that are useful for planning. Chapter 3 introduces deep skill chaining (DSC), a skill discovery algorithm in the simplified setting where an agent's task is to reach a pre-defined set of goal states. Chapter 4 discusses the challenges of sequential composition of skills when their subgoal regions are changing over time, as they do during the option discovery process. Chapter 5 discusses the problem of effectively learning *initiation sets* of options, which we define as the region from which option policy execution is likely to be successful. These initiation regions reduce the effective branching factor of the decision making problem, leading to more sample-efficient learning and robust composition.

2. **Option discovery for exploration and planning.** Exploration in RL can be viewed as a drive to learn abstract models of the world and to continually increase the region over which that model is reliable for planning. Chapters 6 and 7 discusses how an agent builds a graph-based abstraction of the world, called a *skill graph*: nodes of the graph correspond to abstract states and edges correspond to option policies. The skill graph incrementally expands towards the frontiers of the agent's experience and supports high-level planning.

3. **Option discovery beyond state-reaching.** Ultimately, exploration (even using options) is intractable if its objective is *coverage*, i.e, a drive to model *all* parts

of the world. Chapter 8 introduces the concept of "proto-goals", the space of all possible goals that could drive the behavior of an agent. The proto-goal RL agent then learns subspaces that are both plausible and desirable, and hence valuable for guiding exploration. Finally, Chapter 9 presents a skill-discovery algorithm that learns the proto-goal space assumed in Chapter 8; the result is a set of abstract binary classifiers that only attend to a subset of state features in high-dimensional observation spaces.

Taken together, this thesis presents algorithmic ingredients for building agents that explore the world with the intention of building abstract, plannable models of the world. This is achieved by first focusing on the option discovery question, then by reasoning about the state abstractions needed to effectively plan with discovered options, and finally, by interleaving the acquisition of state and action abstractions in a never-ending cycle so that the agent can continually increase its competence in the world. But, before diving into novel contributions, we next introduce the necessary background to understand the remainder of this thesis and situate its contributions in related work.

CHAPTER 2

Background and Related Work

Our goal is to create artificially intelligent agents, i.e, agents that possess the "computational part of the ability to achieve goals in the world" (McCarthy, 1997). We will begin by casting this overarching ambition into the reinforcement learning framework (Section 2.3). Then, we will focus on certain properties that an agent must likely possess for effective reward maximization; this will include topics such as exploration, model-based RL, hierarchical RL, option discovery and finally, ways of integrating them all together.

2.1 Reinforcement Learning

Our ambition of creating artificially intelligent agents can be summarized with the *reinforcement learning problem.* The RL problem captures the challenge faced by an agent: it must sense the environment and take actions in it so that it achieves its own goals and purposes. The *reward hypothesis* in RL distills the notion of "achieving goals and purposes" into a the maximization of a scalar signal called the reward (Sutton, 2004; Littman, 2017). This reward maximization is subject to certain constraints: the agent is bounded in memory and computation and must react to changes in the environment in real-time (Silver et al., 2021; Russell and Subramanian, 1994).

The agent in the RL problem can be represented as a policy π that maps observations o_t to actions a_t. The environment on the other hand, is a system that receives the agent's action a_t and returns a new observation o_{t+1} and reward r_t back to the agent; o_{t+1} reflects a partial view of the new internal state s_{t+1} of the environment and r_t represents the feedback on how effective the agent's action a_t was at time t. Anything outside the computational process that produces actions (the agent) can be thought of as the environment. This agent-environment boundary is often counter-intuitive; for example, the program running inside a robot's operating system that computes the next action is considered to be the agent, while its own body (for example, the sensors and actuators) is part of the environment. The reward r_t is a real-valued scalar that captures the "goals and purposes" of that agent (Sutton, 2004; Littman, 2017); for example, a PhD candidate about to graduate could get a small negative reward for every day that they do not submit their thesis.

Formally, the RL problem can be summarized as finding a policy π^* that maximizes the expected cumulative sum of rewards:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}\left[\sum_t r_t\right]. \tag{2.1}$$

2.1.1 Markov Decision Processes

In this thesis, we make two simplifying assumptions about the RL problem. First, we assume that time ticks in discrete units, even though in reality, time evolves continuously. Second, we assume that the agent can sense the underlying state of the environment, i.e, the current observation has all the information the agent needs to make an optimal decision. This is a strong and unrealistic assumption: in the natural world, it would be impossible for an agent to perceive the entire state of the world (maybe expressed as the position and velocity of every particle). Even in many simpler problems of interest, partial observability is ubiquitous and information that is key to good decision-making may be unavailable to the agent. Nevertheless, many problems in the world do afford the agent the privilege of observing all relevant aspects of the environment, and we will limit our discussion of abstraction discovery to such problems. As a result of the second assumption, we will use the terms observation and state interchangeably.

A consequence of these simplifying assumptions is that we can limit space of environments to those described as Markov Decision Processes (Puterman, 1994). An MDP models discrete-time sequential decision-making problems composed of five quantities $M = (S, A, R, T, \gamma)$. S is the state-space, describing the possible configurations of the world. A is the action space, the set of decisions the agent has to choose between. $R : S \times A \to \mathbb{R}$ is the reward function that maps states and actions to a real-valued scalar quantity. $T : S \times A \to \Delta(S)$ is the transition function that maps states and actions to a distribution over next states. Finally $\gamma \in [0, 1]$ is the discount factor that encodes the preference for near-term rewards over longer-term ones.

Central to the MDP is the Markov assumption which indicates that the transition function and reward function both only depend on the current state and action, and not the entire history:

$$\begin{aligned} T(s_{t+1}|s_t, a_t) = Pr(s_{t+1}|s_0, a_0, ..., s_t, a_t) &= Pr(s_{t+1}|s_t, a_t), \\ R(s_{t+1}|s_0, a_0, ..., s_t, a_t) &= R(s_t, a_t, s_{t+1}). \end{aligned}$$

The agent interacts with the environment via its **policy** $\pi : S \to A$, which prescribes an action for every state in the MDP. When the policy is stochastic, the agent samples from the probability distribution over actions $a \sim \pi(a|s)$.

The value function $V^\pi : S \to \mathbb{R}$ describes the expected cumulative reward when following policy π from state s:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}\left[R(s,a) + \gamma \mathbb{E}_{s' \in S}[V^{\pi}(s')]\right] \\ &= \sum_{a \in A} \pi(a|s) \sum_{s' \in S} \left[R(s,a,s') + \gamma T(s'|s,a) V^{\pi}(s')\right]. \end{aligned}$$

The action-value function $Q^{\pi} : S \times A \to \mathbb{R}$ describes the expected cumulative reward obtained from taking action a from state s and then following policy π thereafter:

$$Q^{\pi}(s,a) = \sum_{s' \in S} \left[R(s,a,s') + \gamma T(s'|s,a) V^{\pi}(s')\right].$$

We will represent the value function and the action-value function of the optimal policy as V^* and Q^* respectively:

$$V^*(s) = \max_a \Big(\sum_{s'} R(s,a,s') + \gamma T(s'|s,a) V^*(s') \Big).$$

2.2 Planning

When the transition and reward functions are known to the agent, it can use planning to find a policy. Here we discuss planning via dynamic programming and refer readers to other sources (for example, Ghallab et al. (2016) and Russell and Norvig (2020)) to learn about other approaches.

Policy Evaluation. The policy evaluation problem can be described as the following question: given a policy π, how much reward does the agent expect to accumulate from following it from a state s? This question can be answered by iteratively applying the Bellman Expectation Equations:

$$V^{\pi}_{k+1}(s_t) = \mathbb{E}_{\pi}[r_{t+1} + \gamma V_k(s_{t+1})] = \sum_a \pi(a|s) \sum_{s',r} Pr(s',r|s,a)\Big[r + \gamma V_k(s')\Big]. \tag{2.2}$$

The fixed-point of this iteration is the true value function of the policy π.

Policy Improvement. Having evaluated the value of a policy, we now turn our attention to policy improvement, which is the process by which we can derive a new policy π' from an old policy π by making it greedy with respect to the value function of the old policy:

$$\pi'(s) = \arg\max_{a \in A} \sum_{s',r} Pr(s',r|s,a)\Big[r + V^{\pi}(s')\Big], \tag{2.3}$$

where $Pr(s',r|s,a)$ is a shorthand for the combination of the next-state distribution according to $T(s'|s,a)$ and the reward function $R(s,a,s')$.

Policy Iteration. We can embed the evaluation of a policy π (Equation 2.2) and its improvement to a new policy π' (Equation 2.3) into a loop called the *policy iteration* loop, which terminates upon finding the fixed-point, the optimal policy π^* in the MDP.

Value Iteration. Every iteration of policy iteration fully evaluates the current policy and then improves it to a new policy. Full policy evaluation however, might not be necessary. Value iteration does a single iteration of policy evaluation followed by policy improvement. Equivalently, value iteration can be thought of as turning the Bellman optimality equation into an iterative update rule:

$$V_{k+1}(s) = \max_a \mathbb{E}[r + \gamma V_k(s_{t'})] = \max_a \sum_{s',r} Pr(s',r|s,a)\Big[r + \gamma V_k(s')\Big], \forall s \in S. \tag{2.4}$$

The sequence of value functions V_k converges to the fixed-point at the optimal value function V^* (V_0 can be initialized arbitrarily for this convergence).

Generalized Policy Iteration. Full sweeps of policy evaluation or improvement can be computationally infeasible in large MDPs. Generalized Policy Iteration (GPI) refers to the family of algorithms in which we interleave partial policy evaluation and iteration until the value function and policy stops changing, which must happen at V^* and π^* respectively. In other words, as long as we keep improving the value function and policy, no matter how incremental, we can still arrive at the optimal quantities of interest.

2.3 Learning from Interaction

So far, we have seen how planning via dynamic programming can be used to convert the transition and reward models into a policy. However, RL agents do not typically have access to such models *a priori*; instead they must learn how to maximize rewards by simply interacting with the environment. This interaction produces a data stream of the form $(s_0, a_0, r_0, s_1, ..)$, and RL solution methods roughly fall into two categories: model-free and model-based.

2.3.1 Model-free RL

It is possible to learn the optimal policy in an MDP without having to access to the transition and reward functions or without even trying to approximate them. These methods are known as *model-free methods*.

Temporal Difference Learning. Temporal difference (TD) learning (Sutton, 1988) leverages observed transitions (s_t, a_t, r_t, s_{t+1}) to update the value function V^π of a policy π. The TD error is defined as:

$$\delta_t = r_{t+1} + \gamma V(s_{t+1}) - V(s_t), \tag{2.5}$$

where the term $r_{t+1} + \gamma V(s_{t+1})$ is a sample-based estimate of the full expectation in the Bellman equation. In other words, rather than computing the expected value over all possible next states and rewards, the agent uses the actual reward and the next state's value observed from the environment. The value function is then updated via:

$$V^{\pi}(s) \leftarrow V^{\pi}(s) + \alpha \delta_t, \tag{2.6}$$

where $\alpha \in (0, 1]$ is a step-size parameter.

Q-learning. TD learning can be used for off-policy control via the Q-learning algorithm (Watkins and Dayan, 1992), which can be elegantly described using the following update rule for the action-value function:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \Big]. \tag{2.7}$$

If all states and actions continue to be updated, then Q-learning recovers the optimal action-value function Q^*, from which the optimal policy can be derived: $\pi(s) = \arg\max_a Q(s, a)$. For Q-learning to be effective, usually some exploration is needed: the agent must occasionally pick exploratory actions, for example, by selecting an action uniformly at random (called ϵ-greedy action-selection).

Deep Q-learning. Watkins and Dayan (1992)'s Q-learning algorithm was developed for the tabular case. When there are too many states (and actions) to fit in a table, we must resort to function approximation. Mnih et al. (2015) introduced the Deep Q-learning algorithm where the agent's Q-function is represented using a neural network θ called the Deep Q-Network (DQN). The Q-function is updated in using transitions sampled from

FIFO buffer (called a replay buffer (Lin, 1993)) using gradient descent on the following loss function:

$$L(\theta) = \mathbb{E}_{\pi}[(y_t - Q_{\theta}(s_t, a_t)^2],$$

where the regression target y_t can be written as:

$$y_t = r_t + \gamma \max_{a_{t+1}} Q_{\theta'}(s_{t+1}, a_{t+1}), \tag{2.8}$$

where θ' is known as a target network and it is a stale copy of θ that is updated periodically. DQN has been augmented with several algorithmic improvements; for example, double Q-learning (van Hasselt et al., 2016), prioritized experience replay (Schaul et al., 2016), recurrent networks (Hausknecht and Stone, 2015; Kapturowski et al., 2019) and distributed architectures with asynchronous acting and learning (Kapturowski et al., 2019).

2.3.2 Model-based RL and Exploration

Model-based RL (MBRL) algorithms use the interaction data to first approximate the transition and reward functions of the MDP, which can be done using maximum likelihood supervised learning to yield an approximate model $\hat{T}, \hat{R}$. This model is then turned into a policy (and optionally a value function) using planning (possibly in combination with model-free updates Sutton 1991). The agent can use this policy (perhaps augmented with ϵ-greedy exploration) to pick actions, collect new data and subsequently improve its model. This approach, while simple, suffers from some drawbacks.

The One-Step Trap and the Simulation Lemma

When a model is learned from finite data, some approximation error is unavoidable. How does this model error relate to the quality of decisions made by the MBRL agent? This question is answered precisely by a foundational result in MBRL, the Simulation

Lemma (Kearns and Singh, 2002):

$$\forall s, \pi : |\hat{V}_s - V_s| \leq \frac{\epsilon_R}{1-\gamma} + \frac{\gamma \epsilon_T}{(1-\gamma)^2},$$

where ϵ_R is error in approximating the one-step reward function and ϵ_T is the error in approximating the one-step transition function. The simulation lemma captures the challenge of doing one-step MBRL: the further we try to look ahead with our model (as signified by a γ closer to 1), the worse our value approximation error becomes (as $\gamma \to 1, |\hat{V} - V| \to \infty$). This result was later extended by Asadi et al. (2018) to the function approximation case: not only does the value error depend on the timescale of the problem, but it also depends on the expressivity of the function class used to approximate the model. Taken together, these results tell us that the depth of planning is severely constrained based on the one-step approximation error and the capacity of our function approximators (Jiang et al., 2016; Talvitie, 2014). The challenge of effective one-step MBRL is evocatively termed as the "one-step trap" by Sutton (2016).

Exploration for Model-based RL

Given that model-prediction error is unavoidable, it seems that the agent is doomed to make exponentially bad decisions as the timescale of the problem increases. One imperative resolution to this conundrum comes from *exploration*: even if the agent has infinite capacity ("perfect" function approximators when the environment is not tabular), it can only reduce model prediction error by collecting more data. But collecting additional data is expensive because it involves interaction with the environment, so what data should the agent preferentially collect so that it can maximally bound the regret it has about its decisions?

Broadly, there are two classes of algorithms that seek to answer this question precisely: Thompson sampling (TS) and optimism in the face of uncertainty (OFU). TS approaches maintain a belief distribution about the model, sample from that distribution and take actions that are optimal under the sampled model (Thompson, 1933; Osband et al., 2013). However, maintaining a Bayesian posterior over the space of models is intractable in large spaces (Strens, 2000), so we do not discuss TS approaches further. OFU approaches like E^3 (Kearns and Singh, 2002), R-Max (Brafman and Tennenholtz, 2002) and MBIE-EB (Strehl and Littman, 2008) first quantify the uncertainty about the agent's approximate model ($\hat{T}$ and $\hat{R}$) and then propose to collect data in parts of the state-action space where that uncertainty is highest. More specifically, the agent is attracted to states in proportion to $\frac{1}{\sqrt{N[s,a]}}$ where $N[s,a]$ is the number of times the agent has taken action a from state s and serves as a proxy for model uncertainty. This causes the agent to preferentially visit parts of the environment where its model is weakest; the result is that the agent can use its one-step model to quickly recover a near-optimal policy.

Exploration in Deep RL

The techniques we discussed for model-based exploration apply only to tabular domains or to continuous MDPs with metric structure and low dimensionality (Kakade et al., 2003). When the observation space becomes large and complicated, tabular approaches do not suffice. For instance, in continuous spaces, an agent may never visit the same state twice, so count-based approaches do not scale (or are not even necessarily well-defined). However, the notion of counts can be generalized to *pseudocounts* (Bellemare et al., 2016): similar states have similar pseudocounts; so, even if the agent visits a state slightly different from one that it has visited before, it will increase that state's pseudocount. Bellemare et al. (2016) introduce a custom tree-based model that estimates the probability density of a state in Atari 2600 games with image-based observations (Bellemare et al., 2013). The CTS model was improved upon by a neural network, Pixel CNN, which approximates probability densities of image-based states without using the domain knowledge required by the CTS model (Ostrovski et al., 2017). To convert probability density estimates into pseudocounts, both papers place severe restrictions on the types of function approximators that can be used, precluding most powerful generative models in deep learning. Recently, Coin Flip Networks (CFN) (Lobel et al., 2023) simplify the problem of pseudocount estimation by obviating the need for density estimation; the resulting approach directly approximates the novelty of a state via its pseudocount $r^{\text{novelty}}(s) = 1/\sqrt{N_\phi(s)}$ and uses that as an exploration bonus for model-free RL (Sutton, 1990; Taïga et al., 2019). Model-based approaches for leveraging the exploration bonus also exist (e.g, Pathak et al., 2017), but they tend to empirically under-perform model-free approaches (Burda et al., 2019a,b; Taïga et al., 2019; Linke et al., 2019).

Challenges of Scaling Novelty Driven Exploration. The success of pure novelty (or surprise) driven exploration (Badia et al., 2020b,a; Kapturowski et al., 2022) has been limited to simple, simulated tasks such as video games (Berseth et al., 2021). Although these environments have high-dimensional state and action spaces, most aspects are usually under the agent's control, mistakes have little to no consequence, and eventually most states are reachable under a reasonably competent policy. However, vast domains like the natural world are characterized by the opposite: most aspects are not directly under the agent's control, there are serious safety considerations, and coverage of the state-space is intractable. To effectively explore in such environments, novelty or surprise seeking behaviors must be carefully paired with considerations such as the bounded capacity of the agent (Russell and Subramanian, 1994), safety (Garcia and Fernández, 2012) and, the topic of this thesis, abstractions—the agent should parsimoniously collect data relevant towards building task-specific representations (Tenenbaum, 2018).

2.4 General Value Functions

Value functions are predictions about the future stream of rewards from a particular state. However, the reward is only one of many signals and by predicting them, the agent can build rich knowledge about the world. Value functions that predict quantities other than the reward are known as *General Value Functions* (GVF) and the signals that they predict are called *cumulants*.

To formally define a GVF, we need to first define the following quantities:

- A cumulant $c : s \to \mathbb{R}$ is a function that maps a state to a scalar and it represents a signal in the agent's data stream that we would like to predict.
- A termination function $\beta : s \to [0, 1]$ represents the probability with which the accumulation of the prediction stops. Put another way, it represents the timescale over which we wish for the agent to predict the cumulant c.
- A stopping function $z : S \to \mathbb{R}$ outputs the stopping value that is added to the accumulation of the cumulant once it terminates.

Put together, the general value function v defines the expected sum of discounted sum of a cumulant up to termination plus a stopping value at the time of termination:

$$v(s_t) = \mathbb{E}_{\pi,\beta}\left[\sum_{k=0}^{K-1} \gamma^k c_{t+k} + \gamma^K z(s_{t+K})\right], \tag{2.9}$$

where the expectation is taken with respect to actions taken by a policy π up to a stopping time determined by the termination function β.

General value functions have been successfully used in deep RL to shape the representations learned by the agent: by predicting signals that are part of the data stream, the agent learns policies that earn more reward in the main task of interest (Jaderberg et al., 2017; Lyle et al., 2021). These works typically assume that the auxiliary tasks are given to the agent, with some rare exceptions (Veeriah et al., 2019).

2.5 The Options Framework

Time in an MDP ticks at every single step. An action at time t affects the state and reward at time $t + 1$ and there is no notion of courses of action persisting over extended, variable periods of time. As a result, most solution methods cannot exploit the benefits of higher-levels of temporal abstraction such as simpler models and faster credit assignment. Such temporal abstraction can be achieved in RL via *options* (Sutton et al., 1999). An option o is a temporally extended action and can be described using a triple:

$$o = (\mathcal{I}_o, \pi_o, \beta_o),$$

where $\mathcal{I}_o : s \rightarrow [0, 1]$, the initiation function, denotes the probability with which the option can be executed at state s; $\beta_o : s \rightarrow [0, 1]$, the termination function, is the probability with which option execution ends in state s; finally, $\pi_o : s \rightarrow A$, the option policy maps states to primitive actions.[1]

When options are given to the agent, they can improve the sample-efficiency of learning. This is mainly due to three reasons. First, an agent with options can explore the environment with greater effectiveness; this is because exploration with primitive actions has the disadvantage that it has a rapidly diminishing probability of deviating from the agent's currently best known policy. Second, options can unlock more effective model-based RL by greatly simplifying the model-learning problem because the agent need only learn the effects of executing options, rather than of the fine-grained details of each low-level action. And third, options reduce the diameter of the MDP–which is the number of steps needed to go between any pair of states–accelerating both exploration and credit assignment.

2.5.1 Option Discovery

Well designed options are clearly useful, but poorly designed options can severely degrade performance (Jong et al., 2008). This places onerous requirements on the programmer to design options suited to each specific task, which is cumbersome and unscalable. The option discovery problem is that of finding a useful set of options

---

[1] Options can also invoke other options, but we simply consider single level hierarchies.

autonomously via interaction with the environment. The discovery problem involves finding all three elements of the option triple, but typically begins by identifying the *subgoal* that the option is supposed to achieve.

Subgoal Options

Technically, an option is simply described by a way of initiating, a way of acting, and a way of terminating—its behaviour need not maximize any objective at all. As an example, consider an option that initiates everywhere, terminates nowhere, and whose policy arbitrarily maps each state to an action; this option does not optimize any useful objective, but perfectly complies with the classical definition of an option (Sutton et al., 1999). However, for option discovery, rather than searching for these three quantities in their raw form, it is often more convenient to think of options as achieving *subgoals*. In fact, the vast majority of the literature on option discovery can be seen as achieving subgoals (e.g., Mcgovern, 2002; Precup, 2001; Colas et al., 2022; Sutton et al., 2024); we refer to these options as *subgoal options* and define them using the following components:

1. **Initiation Set.** The initiation set of the option is the set of states from where the option has a high probability of achieving its subgoal.
2. **Policy.** The option policy maps states to actions, and is optimized to reach the option's subgoal; this policy can be trained by converting the subgoal into a (pseudo-) reward function.
3. **Termination function.** Option execution must terminate in those states where the option's subgoal is achieved.

Subgoal options are a specific type of option, i.e., all subgoal options are options, but not all options are subgoal options.

This subgoal is used to specify the *pseudo-reward function* R_o (also called a *subgoal reward function*, or the *option reward function*) and the termination function β_o. This can be done in the same way as in the GVF formulation in Equation 2.9: the option accumulates a pseudo-reward at every time step that it executes, and it finally terminates with a stopping value that depends on the current state. Once this pseudo-reward function is identified, any policy learning technique can be used to maximize it.

In the remainder of this thesis, the subgoal of the option is always encoded as a binary function of state and the option reward function takes one of two forms. Sometimes, it chosen to induce *shortest path* options, meaning that the option gets a pseudo-reward of 0 for every time step except for when it terminates in a state that achieves its subgoal, getting a reward of 1. Other times, the option reward function is chosen to be *reward respecting*: the option accumulates task rewards until it terminates, when it is given a stopping reward based on whether the subgoal was achieved (Sutton et al., 2024). In short, $R_o(s,a,s') = \beta_o(s') = 1$ when the subgoal is achieved, and $R_o(s,a,s') = 0$ or $R_o(s,a,s') = R(s,a,s')$ and $\beta_o(s') = 1-\gamma$ otherwise. In subsequent chapters, we exclusively consider the discovery of subgoal options and overload the subgoal of the option with its termination function.

2.5.2 Connection between Options and GVFs

We discussed GVFs as functions that *predict* signals other than the reward. However, it is also possible to think of GVFs from the perspective of *control*: rather than simply predicting a cumulant, the agent can learn a policy to maximize it. This policy is exactly the option policy π_o and termination function of the option is exactly the termination function in the GVF discussion. An option is learned to achieve a *subgoal*, which is described using the cumulant (and stopping function) and a timescale at which it must be achieved using the termination function.

GVFs provide a framework to mathematically reason about signals other than reward that the agent could aim to maximize; then tools like generalized policy iteration tell us how to maximize such signals. But given a rich data stream, which of the many possible signals *should* the agent choose to maximize? This is the problem of option discovery in the language of GVFs.

2.6 Goal-conditioned RL

When using function approximation, it is natural to parameterize each GVF separately, as in the *Horde* architecture (Sutton et al., 2011). But in the same way that the state-space has structure that function approximation seeks to exploit, the goal-space too has structure. For example, the GVF for navigating to a particular target location is similar to the GVF for reaching a different target location nearby in Euclidean distance to the first one. Universal Value Function Approximators (UVFAs) (Schaul et al., 2015) exploit this structure to represent all GVFs using the same function approximator; the result is a value function and a policy that, in addition to taking in state, also accept a goal input, i.e, $V_\theta : S \times G \to \mathbb{R}$ and $\pi : S \times G \to A$.

A goal can be formally defined using a triple: $g \in G = (e_g, R_g, \gamma_g)$, where $e_g : S \to \mathbb{R}^d$ is a goal-description vector that can for example, be used to condition a policy $\pi(s|e_g)$ or a value function $V(s|e_g)$; $R_g : S \to \mathbb{R}$ is a goal reward function that maps each state to real-valued number and finally $\gamma_g : S \times A \times S \to [0, 1]$ describes the goal's continuation function, and hence the timescale for achieving that goal. The goal description vector can be states that the agent wants to reach (Andrychowicz et al., 2017), target locations in navigational problems, images describing how the agent would like the world to look (Nair et al., 2018), latent embedding vectors that encode abstract ways of behaving (Eysenbach et al., 2019a) or abstract binary classifiers of state that the agent is motivated to turn on (Bagaria and Schaul, 2023). The goal reward function can be sparse (e.g., 1 when the goal is achieved and 0 otherwise) or dense (e.g., Euclidean distance to some target state). Continuation functions can also take many forms; a common example is when $\gamma_g = 1$ outside some binary subgoal region and 0 inside of it.

2.6.1 Goal-conditioned RL is Closely Linked to Options

An option corresponds to a one-hot goal description vector e_g, i.e., an index into the agent's option set. The goal reward function R_g is equivalent to the option's pseudo-reward function and the goal's continuation function corresponds to the option's termination function. In the same way that the options framework often has a policy over options that maps a state to an option (Bacon et al., 2017), a complete goal-conditioned agent has a higher-level goal policy that maps a state to a goal. In fact, this way of looking at goals also subsumes feudal hierarchies (Dayan and Hinton, 1993), in which managers set goals for workers, either in the form of abstract latent vectors ($e_g \in \mathbb{R}^d$) (Vezhnevets et al., 2017) or as states for the agent to reach ($e_g \in S$) (Nachum et al., 2018; Levy et al., 2019). In all these cases, the higher-level policy operates at a coarser timescale and each high-level decision is followed by a policy execution.

2.6.2 Hindsight Experience Replay

Universal Value Function Approximators (UVFAs) provide a useful parameterization of goal-conditioned value functions $V_g \approx V_\theta(s, g)$, but do not address the problem of sample-efficient learning of such value functions from data. Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) uses reward relabeling to do off-policy learning on goals that are different from the one that was used to generate data. More specifically, suppose an agent observed a trajectory $\tau_g = (s_1, a_1, R_g(s_1), ..., s_T, a_T, R_g(s_T))$ while pursuing goal g. Instead of only using τ_g to update V_g, the agent can relabel τ_g based on some *other* goal (say g') and use that to update $V_{g'}$: $\tau_{g'} = (s_1, a_1, R_{g'}(s_1), ..., s_T, a_T, R_{g'}(s_T))$. A goal-conditioned reward function is typically given to the agent as additional domain knowledge (Schaul et al., 2015; Andrychowicz et al., 2017), with some rare exceptions (for example, the work of Warde-Farley et al. (2019a) and Chapter 9). Any off-policy learning algorithm can be used in conjunction with HER to learn the goal-conditioned value function; the original paper used DQN (Mnih et al., 2015) and DDPG (Lillicrap et al., 2015).

HER provides a way to learn about goals other than the one used to generate the data. But, which of the many goals should the agent learn about in hindsight? These are typically heuristically chosen from the states visited in the trajectory τ_g; common heuristics include random sampling, picking the last state in the trajectory (Andrychowicz et al., 2017), finding a goal that maximizes learning progress proxies (Bagaria and Schaul, 2023) or using inverse RL to find a goal for which τ_g was optimal (Eysenbach et al., 2020).

2.7 Related Work on Option Discovery

New algorithms for option discovery are a core contribution of this thesis. As a result, we now survey this prolific area of research by organizing it into four broad categories: spectral methods, empowerment maximization, learning from extrinsic reward, meta-learning and finally, methods with formal guarantees.

2.7.1 Spectral Methods for Option Discovery

An MDP can be modeled as a graph: nodes represent states and edges represent state adjacency, i.e, two nodes are connected if there exists an action that has non-zero probability of causing a transition between them. Several methods perform a spectral decomposition of the Laplacian (Chung and Graham, 1997) representation of the graph for exploration.[2] Typically, an option is associated with each eigenvector of the graph Laplacian and these options, called *eigen options*, demonstrate strong exploratory properties (Machado, 2019). Importantly, each eigenvector captures different time-scales of diffusion in the environment; eigenvectors with smaller eigenvalues correspond to larger time-scales (Machado et al., 2023). Covering options Jinnai et al. (2019) argue that rather than constructing an option for every eigenvector of the graph Laplacian, a single option

---

[2] Specifically, the Laplacian is defined in matrix form as $\mathcal{L} = D^{-1/2}(D - W)D^{-1/2}$, where W is the weight matrix of the graph and D is the diagonal matrix whose entries are the row sums of W (Machado et al., 2018b).

constructed based on the *Fiedler vector* (Chung and Graham, 1997), is sufficient. This is because that single option minimizes the cover time of underlying MDP, which loosely refers to how long it takes for a random policy over options to visit all states. More data can be collected with the newly added option, the resulting adjacency matrix can be used to learn yet another option, and so on. Klissarov and Machado (2023) recently combined covering options and eigenoptions to propose a new skill discovery algorithm that demonstrates impressive exploration in a variety of reinforcement learning problems.

Laplacian-based methods are closely related to successor representations, which represent the transition dynamics by quantifying the discounted state occupancy of a policy (Dayan, 1993). Successor representations can be computed online using TD methods, can generalize policies quickly to related tasks and can be approximated in high-dimensional spaces using neural networks (Barreto et al., 2017; Borsa et al., 2019). Furthermore, the Option Keyboard algorithm leverages successor representations to compose options *within* time: given that an agent has options $o_1, ..., o_K$, it can select an action at the current timestep using a mixture of its option policies. This results in a large and rich space of behaviors; importantly, the resulting combination of policies has policy improvement guarantees (Barreto et al., 2019a). In fact, Machado et al. (2023) provide an integrated agent architecture in which successor representations be used for discovering options, which can then be combined using option keyboard, which in turn results in higher quality data for improving the agent's successor representations, and so on.

Another interesting connection is between the successor representations and proto-value functions (PVFs) (Mahadevan and Maggioni, 2007), which are a basis for all value functions in an MDP; proto-value functions are also the eigenvectors of the graph Laplacian (Machado et al., 2023), implying that (approximate) PVFs are used as intrinsic reward functions while learning eigenoption policies. Representation learning using successor features is an exciting frontier of research (Farebrother et al., 2023; Touati and Ollivier, 2021; Stachenfeld et al., 2017; Carvalho et al., 2022), but further discussion is beyond the scope of this thesis.

Successor representations and Covering Eigenoptions are closely related to the work in Chapters 6,7, and 9 of this thesis. Instead of starting from the graph Laplacian, we use approximate state visitation counts (Bellemare et al., 2016) as guidance for option discovery. Formally, the norm of the successor representation are equivalent to state visitation counts (Machado et al., 2019). Informally, eigenoption-type methods and count-based methods both roughly capture the same intuitive idea of the agent's frontier: both try to identify states that are currently difficult (but possible) to reach, and then construct options to make them easier to reach in the future. In this thesis, we rely on count-based methods because they have been successfully scaled to large environments with impressive results (Badia et al., 2020b,a; Kapturowski et al., 2022; Silver et al., 2017a). But, as the approximation techniques for Laplacian-based methods improve (Wu et al., 2019; Wang et al., 2021; Touati and Ollivier, 2021), we could imagine porting the lessons from this thesis to those methods as well. In other words, the core ideas of this thesis (for example, how to interleave exploration and planning in a single agent) are complementary to successor- based option discovery methods.

2.7.2 Empowerment based Skill Discovery

Empowerment quantifies the degree of control an agent has over its environment (Volpi and Polani, 2023). Intuitively, there are many interpretations and uses of empowerment: an intrinsic drive to reach states that have the highest number of states reachable from them (Klyubin et al., 2005), to maximize social influence in multi-agent settings (Jaques et al., 2019), or to seek agreement between future states and the agent's internal representations (Hafner et al., 2020). Mathematically, empowerment is the mutual information between an agent's actions and future states (Klyubin et al., 2008). Computationally, this mutual information can be estimated using tools from variational inference (Mohamed and Jimenez Rezende, 2015).

The variational approximation derived by Mohamed and Jimenez Rezende (2015) searched for an open-loop K-step sequence of actions that maximized empowerment. Since then, methods have been proposed to discover closed-loop policies, i.e., skills or options. Different skills are parameterized via context vectors z drawn[3] either from a fixed, simple distribution such as a Gaussian distribution G or from a learned distribution $p_\psi(z)$ (Levy et al., 2023). Then, the following functions are learned: a skill-conditioned policy $\pi_\theta(: |s, z)$ that produces trajectories τ, a classifier q_ϕ that predicts which skill was executed given τ and a distribution over skills $p_\psi(z)$. The training objective J, which is analogous (Achiam et al., 2018) to β-VAEs (Higgins et al., 2017), seeks to maximize the entropy of skill policies while encouraging different skills to lead to diverse trajectories:

$$J(\theta, \phi) = \max_{\theta,\phi,\psi} \mathbb{E}_{z\sim p_\psi(z), s\sim p_\theta(s|z)} \Big[\mathbb{E}_{\tau\sim\pi_\theta, z}[\log q_\phi(z|\tau)] + \beta\mathcal{H}(\pi_\theta|z)\Big], \tag{2.10}$$

where the entropy term, $\mathcal{H}(\pi_\theta|z)$, encourages the policy to be as close to the uniform distribution as possible, much like the entropy term in VAEs (Kingma and Welling, 2014). Strouse et al. (2022) noticed that the discriminator, $q_\phi(z|\tau)$ is pessimistic in new states; to address this pessimism, their algorithm, DISDAIN, augments skill learning with a novelty bonus. CIC (Laskin et al., 2022) improves the optimization of the mutual information objective (Equation 2.10) using contrastive learning.

Variational Intrinsic Control (VIC) (Gregor et al., 2017; Kwon, 2020) can be obtained by setting $\beta = 0$ and approximating $q_\phi(z|\tau)$ with $q_\phi(z|s_T)$, where s_T is the final state in trajectory τ. Similarly, DIAYN (Eysenbach et al., 2019a) can be obtained by approximating $\log(q_\phi(z|\tau))$ as the sum of per-timestep log probabilities along the trajectory $\sum_{t=1}^{T} \log(q_\phi(z|\tau))$. Instead of the sum-based decomposition of the trajectory (which treats each transition as being independent from the others), VALOR approximates $q_\phi(z|\tau)$

---

[3] Each context vector z corresponds to an skill, but we use the z notation to be consistent with the empowerment literature. Furthermore, z is usually a continuous vector in $\mathbb{R}^n$, while skills are typically described as a discrete set.

using an LSTM architecture (Achiam et al., 2018). Relative VIC (Baumli et al., 2021) departs from this view slightly by introducing a new term to the optimization: rather than requiring that trajectories be distinguishable from the observed trajectory, they additionally require that the trajectory *not* be distinguishable from the final state alone, thereby encouraging options to cause characteristic *changes* in state, rather than taking the agent to different parts of the state-space alone.

Skill-discovery via empowerment maximization usually proceeds in two stages: a *pre-training* stage, in which skills are learned by maximizing the objective in Equation 2.10 and an *evaluation* phase in which learned skills are composed to solve new tasks. Some algorithms explicitly learn skills that have easily predictable effects (Sharma et al., 2020b,a) and cover the state-space (Campos Camúñez et al., 2020) so that they can be composed easily using search-based procedures (Deng, 2006; Williams et al., 2016) at test-time. The VISR algorithm (Hansen et al., 2020) uses successor features (Dayan, 1993; Borsa et al., 2019; Barreto et al., 2019b) to quickly adapt learned skills to new tasks using a process similar to Option Keyboard (Barreto et al., 2019a).

Connection to goal-based exploration. When the variational distribution in Equation 2.10 is normal and fixed, empowerment objectives reduce to goal-based exploration in RL (Choi et al., 2021), by which we mean methods that propose random target goal states and use a goal-conditioned policy (Schaul et al., 2015) to reach them; for example, Hindsight Experience Replay (Andrychowicz et al., 2017) and Go-Explore (Ecoffet et al., 2019). In fact, it is possible to think of goal-based exploration and variational empowerment as lying on a spectrum: the more expressive the variational distribution, the more powerful, albeit non-stationary, the associated representation learning problem (Choi et al., 2021). Furthermore, DISCERN advocates for taking a mutual information maximization approach to goal-conditioned reward functions (Warde-Farley et al., 2019b), and MEGA argues that empowerment maximization is roughly equivalent to maximizing the size of the set of goals that can be achieved by the agent's policy (Pitis et al., 2020c); these findings further blur the lines between goal-based exploration in RL and empowerment maximization. Seen this way, it is possible to consider the some of the exploration algorithms (Chapters 6,7, and 8) in this thesis as approximate empowerment maximization as well.

Open Challenges. Despite significant progress in online mutual information estimation, Equation 2.10 remains challenging to estimate and optimize (Achiam et al., 2018). Furthermore, the policy over options in most empowerment driven skill discovery papers are fixed (for example, a random policy), with some rare exceptions: HIDIO (Zhang et al., 2021a) uses model-free RL and DADS (Zhang et al., 2021a) uses the cross-entropy technique, but these methods do not do explicit state abstraction, and do not propose an integrated architecture for model-based RL. Nevertheless, empowerment maximization remains a promising objective for skill discovery in large open worlds that demand significant exploration (Du et al., 2023).

2.7.3 Option Discovery from Extrinsic Rewards

Most option discovery techniques are driven by auxilliary objectives, which are sometimes at odds with the agent's overall objective of reward maximization. For example, methods that identify bottleneck states are useful for solving many tasks (Mcgovern, 2002), but are counterproductive in others. Similarly, novelty maximization is useful in some tasks but not others (Taïga et al., 2019). Is there a way to build agents that still maximize reward but have a minimal inductive bias towards temporal abstraction? In other words, rather than specifying a heuristic that should drive the discovery of these options, what if the agent flexibly determined its own auxilliary tasks in a manner that improved its ability to maximize rewards in a complex environment? Indeed, this can be considered a key insight from *Reward is Enough* (Silver et al., 2021): the right type of subtasks will automatically emerge if we build scalable agents that attempt reward maximization in complex environments.[4] This desire to learn options from task rewards has led to several families of techniques: option-critic, feudal networks and meta-gradient driven discovery.

Option Critic Methods

Option-critic methods search for options that maximize overall reward by using the policy gradient theorem (Bacon et al., 2017). The hope behind this approach is that skills would emerge naturally by interacting with the environment, without the need to manually specify what objective should be used to learn them. Arguably, such skills would lead to high performance as they are directly defined in terms of maximizing the expected return.

An issue observed with Option Critic algorithms is that the resulting options tend to degenerate into primitive actions. Harb et al. (2018) derive a regularization term to the objective function that discourages option interruption, i.e, a preference over options that last longer than others. Several contributions to the Option Critic algorithm include learning safe policies (Jain et al., 2018), using second order update rules (Tiwari and Thomas, 2019), learning all options simultaneously (Klissarov and Precup, 2021), using multiple discount factors (Harutyunyan et al., 2019), learning initiation sets (Khetarpal and Precup, 2019), learning option interruptions in an off-policy manner (Harutyunyan et al., 2019) or dealing with continuous action spaces (Klissarov et al., 2017).

These several contributions have significantly improved the Option-Critic algorithm, making it more generally applicable with better performance. The end-to-end nature of Option Critic makes it scale gracefully to large problems with high-dimensional observation spaces. But, since all learning stems from the extrinsic reward function, option-critic methods do not result in sample-efficient learning in sparse reward problems. Furthermore, the problem of how to prevent options from collapsing to primitive actions, although

---

[4] Of course, it is possible that we do not need to explicitly program a bias towards temporal abstractions, and that itself could be an emergent property of reward maximization.

mitigated, has not been entirely resolved; neither has the related problem of finding solutions with Option Critic that are composed entirely of a single option execution. Finally, how resulting options can be used for planning or be transferred across contexts remains unclear.

Feudal Approaches to Skill Discovery

In feudal reinforcement learning (Dayan and Hinton, 1993), the agent's policy is decomposed hierarchically into *managers* and *workers*: managers set subgoals for workers to achieve and workers use flat RL to achieve those subgoals. In this way, goal-setting is decoupled from goal-achievement; each level in the hierarchy communicates to the level below it what must be achieved, but does not specify how to do so. This is a form of temporal abstraction in which higher levels of the hierarchy make decisions at a lower temporal resolution than lower levels.

Feudal RL was extended to deep RL by Feudal Networks (FuN) (Vezhnevets et al., 2017). FuN learns a two-level hierarchy in which the higher level manager outputs a latent vector that specifies the direction in which the lower level worker should modify the agent's current state. The manager policy is trained using policy gradients on the task reward function, while the worker policy is trained only using its intrinsic subgoal/pseudo reward function. As with most skill discovery methods, the high-level policy is trained at the same time as the low-level policy; as the low-level policy changes during learning, a data from a high-level action taken in the past may not yield the same low-level behavior in the future (Nachum et al., 2018). This non-stationarity was noticed and addressed using relabeling tricks and off-policy learning in the HIRO algorithm (Nachum et al., 2018). Later, Hierarchical Actor Critic (Levy et al., 2019) improved upon HIRO by removing the need for dense reward functions, by instead using hindsight experience replay (Andrychowicz et al., 2017) and stably training multi-level hierarchies (Li et al., 2019). Like Option-Critic, these methods relied on extrinsic rewards to drive the learning of the high-level policy; if the high-level policy is poorly trained, it outputs goals that fail to drive the learning progress of the lower-level policy. To address the exploration challenge, recent methods like HAC-Explore incorporate an novelty-based intrinsic rewards (McClinton et al., 2021; Röder et al., 2020) or demonstrations (Gupta et al., 2019) to solve longer horizon tasks. HIRO and HAC demonstrated stronger empirical performance on continuous control tasks, but they made assumptions that limited their applicability to feature-based observation spaces (as opposed to more generic observations like images, which could be dealt with by FuN).

Another challenge faced by feudal methods such as HIRO and HAC is that the action-space of the higher-level policy is the entire state-space of the environment. The larger action-space further complicates the learning of the high-level policy. As a result, several papers constrain the output space of the high-level policy using domain knowledge (for example, the coordinates of the center of mass in robot navigation). This problem has since been formally studied through the lens of information theory: can we find a representation of state that can be used as the goal-space, but results in bounded value loss? Nachum et al. (2019) use mutual information estimation using contrastive predictive coding (van den Oord et al., 2018) to find such a representation space. On the other hand, DISCERN (Warde-Farley et al., 2019a) uses the mutual information between the subgoal state and the state reached after policy execution as the reward function for training the low-level policy. These approaches blur the lines between reward-driven option discovery and empowerment techniques, that also maximize the mutual information between a skill policy and the state that results from its execution.

2.7.4 Discovery using Meta-Learning

A major benefit of learning options is that of reuse: options learned in one part of the state-space could speed up learning in another (Taylor and Stone, 2009; Konidaris and Barto, 2007). For example, the ability to walk, once acquired, can be transferred to many different contexts. Some methods have tried to discover transferable options using meta learning. MLSH (Frans et al., 2018) discovers a set of policies and trains them in a way that will make them reusable across a pre-specified task distribution (Nam et al., 2022; Gupta et al., 2018; Fu et al., 2023). MODAC (Veeriah et al., 2021) uses meta-gradients (**?**Oh et al., 2020) to do the same: an outer loop learns option pseudo-reward and termination functions that an inner loop maximizes using traditional RL methods like policy gradients; the outer-loop of the optimization proceeds based on reward maximization.

Meta-learning approaches have also sought to address the exploration question in non-stationary and multi-task settings. When the agent finds itself in a new environment, can it leverage its past experiences to targetedly explore this new environment? This problem is called *meta-exploration* in Beck et al. (2023). For example, when someone is in a new house and they have to look for utensils, they begin their search from the kitchen; similarly, we would like to create RL agents that can direct their exploration for quick adaptation in new environments (Gupta et al., 2018). This is the motivation of the Adaptive Agent (AdA) (Bauer et al., 2023) that uses meta-learning to train a policy capable of quick adaptation in a massive task space (OEL Team et al., 2021). In AdA, a higher-level policy uses curriculum learning approaches (such as prioritized level replay (Jiang et al., 2021)) to pick tasks for a lower-level policy to achieve. Each sampled task, which is equivalent to an option, is attempted by the lower-level policy for several trials, until a new task is sampled by the high-level policy. A sequence-based model is used to meta-learn across all the trials in that task; when a new task is sampled, the agent's memory is reset. The result is a policy that learns how to explore in partially observable environments: given a new, unseen task, the policy explores the uncertain parts of the environment in earlier trials to exploit in later ones. AdA, and similar methods, do not strictly "discover" the task/option space themselves: the space of possible tasks/options is defined in the XLand simulator (OEL Team et al., 2021); however, this space has 10^{40} tasks, and if the agent randomly sampled tasks from that space, it would be unlikely to result in any learning progress. So, even though curriculum learning in such a vast task/option space does not strictly satisfy the definition of discovery, it provides important lessons for scaling option discovery to vast environments that benefit from open-ended learning.

Open Challenges. Meta-learning approaches have demonstrated abilities of transfer, adaptation and meta-exploration—abilities that have been challenging to acquire using other techniques. Furthermore, meta-learning via in-context learning (Dong et al., 2022; Bauer et al., 2023; Raparthy et al., 2023) provides a scalable, and potentially simpler way, to acquire these crucial capabilities. Existing meta-learning approaches rely on a specialized multi-task formulation, with clear task boundaries and episodes. Methods that lift these assumptions will be able to bring these capabilities to the more generic single-task formulation: the environment is described as a single decision process, and the agent flexibly proposes and solves subtasks so that it may maximize rewards—neither requiring explicit task boundaries (Humplik et al., 2019) nor a re-settable simulator (Xu et al., 2020).

2.7.5 Option Discovery with Formal Guarantees

Many of the option discovery methods that we have discussed so far rely on *proxy* objectives; these objectives include finding bottleneck states, empowerment maximization, more reliable composability, and so on. The intuition is that if the agent had options that maximized these proxy objectives, it would unlock agent-level capabilities such as effective exploration, credit assignment, or transfer. Indeed, these methods often show empirical success in some scenarios, but the formal connection between these proxy objectives and the overall objectives of the agent is unclear (Solway et al., 2014). For example, options that target "bottleneck" states are empirically useful in some navigation tasks, but what kind of performance can we expect from the same technique in an entirely different problem? In fact, several papers have shown that not all skills are created equal—that is, options that are perfectly suited for a particular task, might severely hurt agent-level objectives in other tasks (Jong et al., 2008; Solway et al., 2014). To address this gap, a class of methods—initiated by Solway et al. (2014)—has sought to discover options with precise guarantees on agent-level objectives. These methods explicitly state the performance criterion of the agent and then derive an algorithm that discovers options with bounded loss on that criterion.

Planning. In the context of planning, the performance criterion of the agent can be stated as minimizing *planning time*, which is the number of iterations of a planning algorithm (like value iteration) needed to converge to the optimal value function. The goal of option discovery in planning can be seen as finding a set of options that minimizes planning time. Jinnai et al. (2018) prove that this problem is NP-hard, even for deterministic MDPs. They provide approximation algorithms with bounded suboptimality, but these are only valid for tabular MDPs with "point options", which are options that initiate and terminate in a single state respectively. While Jinnai et al. (2018) provides options that bound the worst-case planing time in single-task settings, *Average Options* (Ivanov et al., 2024) minimize the planning time averaged over a distribution of tasks. More precisely, they consider tasks that differ only in terms of their start and goal states and seek to find options that reduce the expected cost of going between *any* two states in the MDP. Unsurprisingly, they find that this problem is also NP-hard but by drawing connections to the k-medians with penalties problem from graph-theory, they introduce approximation algorithms for discovering options that provably minimize planning time in multi-goal MDPs. Planning can also be sped up using options in the single-task setting: Wan and Sutton (2022) present an option discovery algorithm that seeks options that maximize reward–similar to option-critic (Harb et al., 2018)–but reduces the number of options available at different states to reduce planning time.

Exploration. In the context of exploration, Jinnai et al. (2019) formalize the performance criterion of the agent as minimizing the number of steps needed for a policy to visit every state (as a proxy for discovering some unknown reward). They show that this performance criterion is related to the graph-theoretic property of *cover-time*, which measures the number of steps needed by a random walk to visit every edge in a graph. Although finding edges that minimize cover-time in a graph is NP-Hard, Jinnai et al. (2019) provide an approximation algorithm that minimizes an upper-bound on the expected cover-time; they do this by leveraging the graph Laplacian of the state transition matrix. This method was later extended to continuous environments using deep learning-based approximations of the graph Laplacian, further suggesting deep connections to the eigenoptions literature (Machado et al., 2017, 2023; Klissarov and Machado, 2023).

Credit assignment. The problem of evaluating a policy can also be aided using options. Bacon and Precup (2016) formally state this problem as that of finding the set of options that can achieve the fixed-point of the Bellman expectation equations in as few iterations as possible. They find that the problem of discovering options that speed up policy evaluation is equivalent to the problem of finding a pre-conditioning matrix for solving large systems of linear equations; in the same way that good pre-conditioning matrices transform optimization problems into simpler ones, options transform MDPs into those that have dynamics that permit faster policy evaluation. While they do not provide a new option discovery algorithm, all methods that have been proposed for discovering a good preconditioning matrix can be applied to discovering options that aid faster policy evaluation.

Transfer. In the context of transfer, Solway et al. (2014) define the optimal set of options as those that maximize the efficiency with which an agent can learn the optimal policy for other, possibly unseen, set of tasks. They show that in this setting, optimal options are those that maximize Bayesian model evidence under the distribution of tasks that the agent is expected to solve. Specifically, a hierarchy that maximizes model evidence, also provably minimizes the geometric mean of number samples needed to find the optimal policy for any task in the given task distribution. Brunskill and Li (2014) consider a similar formulation of the option transfer problem: given interaction data from a set of tasks, how can an agent learn options that minimize the sample complexity of learning in a future stream of tasks? They find that this problem is at least as hard as the set cover problem in Operations Research, and is thus also NP-Hard. They use a greedy approximation algorithm for option discovery and evaluate it empirically in a tabular MDP.

In summary, bringing formal guarantees into the option discovery pursuit is an important direction of research, but one that is in the early stages of development and which has not yet resulted in many scalable methods. One could imagine integrating several such algorithms into a single agent so that it possesses different options that specialize in planning, exploration and credit assignment. This is different from the approach taken in this thesis, which de-emphasizes theoretical guarantees in favor of empirical scalability, focusing on discovering options that are sequentially composable, useful for exploration and amenable to planning.

2.8 Summary and Conclusions

The RL problem captures our overarching ambition of creating general-purpose agents that can learn to achieve their goals and purposes via interaction with the environment. Model-free RL provides a scalable way to learn behaviors in environments with complex dynamics, but is sample inefficient. By contrast, model-based RL algorithms make better use of data because they leverage the power of planning. However, most model-based algorithms attempt to learn a one-step model of the environment, which is untenable because of the problem of errors compounding over time. Options (and other GVFs) provide a way out of this problem, but useful options are not known *a priori* and the agent has to discover them via interaction. While the option discovery question has often been studied separately from the exploration question in RL, I view them as two sides of the same coin: exploration must be in service of learning an abstract model of the world. Subsequent chapters will shed further light on how options can be discovered in a way that encourages exploration and enables effective planning via model-based RL.

CHAPTER 3

Option Discovery using Deep Skill Chaining

How can agents autonomously construct useful skills via interaction with the environment? While a large body of work has sought to answer this question in small discrete domains, skill discovery in high-dimensional continuous spaces remains an open problem. An early approach to skill discovery in continuous-state environments was skill chaining (Konidaris and Barto, 2009b), where an agent constructs a sequence of options that target a salient event in the MDP (for example, the goal state). The skills are constructed so that successful execution of each option in the chain allows the agent to execute another option, which brings it closer still to its eventual goal. While skill chaining was capable of discovering skills in continuous state spaces, it could only be applied to relatively low-dimensional state-spaces with discrete actions.

We introduce a new algorithm that combines the core insights of skill chaining with recent advances in using non-linear function approximation in reinforcement learning. The new algorithm, *deep skill chaining*, scales to high-dimensional problems with continuous state and action spaces. Through a series of experiments on five challenging domains in the MuJoCo physics simulator (Todorov et al., 2012), we show that deep skill chaining can solve tasks that otherwise cannot be solved by non-hierarchical agents in a reasonable amount of time. Furthermore, the new algorithm outperforms state-of-the-art deep skill discovery algorithms (Bacon et al., 2017; Levy et al., 2019) in these tasks.

3.1 Method

Deep skill chaining (DSC) is based on the intuition that it is easier to solve a long-horizon task from states in the local neighborhood of the goal. This intuition informs the first step of the algorithm: create an option that initiates near the goal and reliably takes the agent to the goal. Once such an option is learned, we create another option whose goal is to take the agent to a state from which it can successfully execute the first option. Skills are chained backward in this fashion until the start state of the MDP lies inside the initiation set of some option. The inductive bias of creating sequentially executable skills guarantees that as long as the agent successfully executes each skill in its chain, it will solve the original task. More formally, skill chaining amounts to learning options such that the termination condition $\beta_{o_i}(s_t)$ of an option o_i is the initiation condition $\mathcal{I}_{o_{i-1}}(s_t)$ of the option that precedes it in its chain.

Our algorithm proceeds as follows: at time t, the policy over options $\pi_{\mathcal{O}} : s_t \in S \rightarrow o \in \mathcal{O}$ determines which option to execute (Section 3.1.2). Control is then handed over to the selected option o_i's internal policy $\pi_{o_i} : s \in S \rightarrow a_t \in \mathbb{R}^{|A|}$. π_{o_i} outputs joint torques until it either reaches its goal ($\beta_{o_i} := \mathcal{I}_{o_{i-1}}$) or times out at its predetermined budget T (Section 3.1.1). At this point, $\pi_{\mathcal{O}}$ chooses another option to execute. If at any point the agent reaches the goal state of the MDP or the initiation condition of a previously learned option, it creates a new option to target such a salient event. The machinery for learning the initiation condition of this new option is described in Section 3.1.3. We now detail the components of our architecture and how they are learned.

3.1.1 Intra-Option Policy

Each option o maintains its own policy $\pi_o : s \rightarrow a_t \in \mathbb{R}^{|A|}$, which is parameterized by its own neural networks θ_o. To train $\pi_o(s; \theta_o)$, we must define o's internal reward function. In sparse reward problems, o is given a subgoal reward when it triggers β_o; otherwise it is given a step penalty. In the dense reward setting, we can compute the distance to the parent option's initiation set classifier and use that to define o's internal reward function. We can now treat learning the intra-option policy (π_o) as a standard RL problem and use an off-the-shelf algorithm to learn this policy. Since in this work we solve tasks with continuous action spaces, we use Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) to learn option policies over real-valued actions.

3.1.2 Policy Over Options

Initially, the policy over options ($\pi_{\mathcal{O}}$) only possesses one option that operates over a single time step ($T = 1$). We call this option the *global option* (o_G) since its initiation condition is true everywhere in the state space and its termination condition is true only at the goal state of the MDP (i.e, $\mathcal{I}_{o_G}(s) = 1 \forall s$ and $\beta_{o_G} = 1_g$). Using o_G, $\pi_{\mathcal{O}}$ can select primitive actions. At first the agent continually calls upon o_G, which uses its internal option policy π_{o_G} to output exactly one primitive action. Once o_G triggers the MDP's goal state N times, DSC creates its first temporally extended option, the *goal option* (o_g), whose termination condition is also set to be the goal state of the MDP, i.e, $\beta_{o_g} = 1_g$.

As the agent discovers new skills, it adds them to its option repertoire and relies on $\pi_{\mathcal{O}}$ to determine which option (including o_G) it must execute at each state. Unlike o_G, learned options will be temporally extended, i.e, they will operate over $T > 1$ time steps. If in state s_t the agent chooses to execute option o_i, then o_i will execute its own closed-loop control policy (for τ steps) until its termination condition is met ($\tau < T$) or it has timed out at $\tau = T$ time steps. At this point, control is handed back to $\pi_{\mathcal{O}}$, which must now choose a new option at state $s_{t+\tau}$.

Option selection: To select an option in state s_t, $\pi_{\mathcal{O}}$ first constructs a set of admissible options given by Equation 3.1. $\pi_{\mathcal{O}}$ then chooses the admissible option that maximizes its option-value function, as shown in Equation 3.2. Since the agent must choose from a discrete set of options at any time, we learn its option-value function using Deep Q-learning (DQN) (Mnih et al., 2015).

$$\mathcal{O}'(s_t) = \{o_i | \mathcal{I}_{o_i}(s_t) = 1 \cap \beta_{o_i}(s_t) = 0, \forall o_i \in \mathcal{O}\} \tag{3.1}$$

$$o_t = \underset{o_i \epsilon \mathcal{O}'(s_t)}{\arg\max}\, Q_\phi(s_t, o_i). \tag{3.2}$$

Learning the option-value function: Given an SMDP transition $(s_t, o_t, r_{t:t+\tau}, s_{t+\tau})$, we update the value of taking option o_t in state s_t according to SMDP Q-learning update (Bradtke and Duff, 1995). Since the agent learns Q-values for different state-option pairs, it may choose to ignore learned options in favor of primitive actions in certain parts of the state-space (in the interest of maximizing its expected future sum of discounted rewards). The Q-value target for learning the weights ϕ of the DQN is given by:

$$y_t = \sum_{t'=t}^{\tau} \gamma^{t'-t} r_{t'} + \gamma^{\tau - t} Q_{\phi'}(s_{t+\tau}, \underset{o' \epsilon \mathcal{O}'(s_{t+\tau})}{\arg\max}\, Q_\phi(s_{t+\tau}, o')). \tag{3.3}$$

Adding new options to the policy over options: Equations 3.1, 3.2 and 3.3 show how we can learn the option-value function and use it for selecting options. However, we must still incrementally add *new* skills to the network during the agent's lifetime. After the agent has learned a new option o's initiation set classifier $\mathcal{I}_o$ (we will discuss how this happens in Section 3.1.3), it performs the following steps before it can add o to its option repertoire:

- To initialize o's internal policy π_o, the parameters of its DDPG (θ_o) are set to the parameters of the global agent's DDPG (θ_{o_G}). Subsequently, their neural networks are trained independently. This provides a good starting point for optimizing π_o, while allowing it to learn sub-problem specific abstractions.

- To begin predicting Q-values for o, we add a new output node to final layer of the DQN parameterizing $\pi_{\mathcal{O}}$.
- We must assign appropriate initial values to $Q_\phi(s, o)$. We follow Konidaris and Barto (2009b) and collect all the transitions that triggered β_o and use the max over these Q-values to optimistically initialize the new output node of our DQN.[1] This is done by setting the bias of this new node, which ensures that the Q-value predictions corresponding to the other options remain unchanged.

3.1.3 Initiation Set Classifier

Central to the idea of learning skills is the ability to learn the set of states from which they can be executed. First, we must learn the initiation set classifier for o_g, the option used to trigger the MDP's goal state. While acting in the environment, the agent's global DDPG will trigger the goal state N times (also referred to as the *gestation period* of the option by Konidaris and Barto (2009b) and Niekum and Barto (2011)). We collect these N successful trajectories, segment the last K states from each trajectory and learn a one-class classifier around the segmented states. Once initialized, it may be necessary to refine the option's initiation set based on its policy. We do so by executing the option and collecting data to train a two-class classifier. States from which option execution was successful are labeled as positive examples. States from which option execution timed out are labeled as negative examples. We continue this process of refining the option's initiation set classifier for a fixed number of episodes, which we call the *initiation period* of the option.

At the end of the initiation period, we fix the option's initiation set classifier and add it to the list of salient events in the MDP. We then construct a new option whose termination

---

[1] Using the mean Q-value is equivalent to performing Monte Carlo rollouts. Instead, we follow the principle of *optimism under uncertainty* (Brafman and Tennenholtz, 2002) to select the max over the Q-values.

condition is the initiation classifier of the option we just learned. We continue adding to our chain of options in this fashion until a learned initiation set classifier contains the start state of the MDP.

3.1.4 Generalizing to Skill Trees

Our discussion so far has been focused on learning skill chains that extend from the goal to the start state of the MDP. However, such a chain is not sufficient if the agent has multiple start states or if we want the agent to learn multiple ways of solving the same problem. To permit such behavior, our algorithm can be used to learn skills that organize more generally in the form of trees (Konidaris and Barto, 2009b; Konidaris et al., 2012). To demonstrate skill trees (and their usefulness), we consider a maze navigation task, E-Maze, with distinct start states in Section 3.2.

3.1.5 Optimality of Discovered Solutions

Each option o's internal policy π_o is is given a subgoal reward only when it triggers its termination condition β_o. As a result, π_o is trained to find the optimal trajectory for entering its own goal region. Naively executing learned skills would thus yield a *recursively optimal* solution to the MDP (Barto and Mahadevan, 2003). However, since the policy over options $\pi_{\mathcal{O}}$ does not see subgoal rewards and is trained using extrinsic rewards only, it can combine learned skills and primitive actions to discover a *flat optimal* solution π^* to the MDP (Barto and Mahadevan, 2003). Indeed, our algorithm allows $\pi_{\mathcal{O}}$ to employ discovered skills to quickly and reliably find feasible paths to the goal, which over time can be refined into optimal solutions. It is worth noting that our ability to recover π^* in the limit is in contrast to feudal methods such as HAC (Levy et al., 2019) in which higher levels of the hierarchy are rewarded for choosing feasible subgoals, not optimal ones.

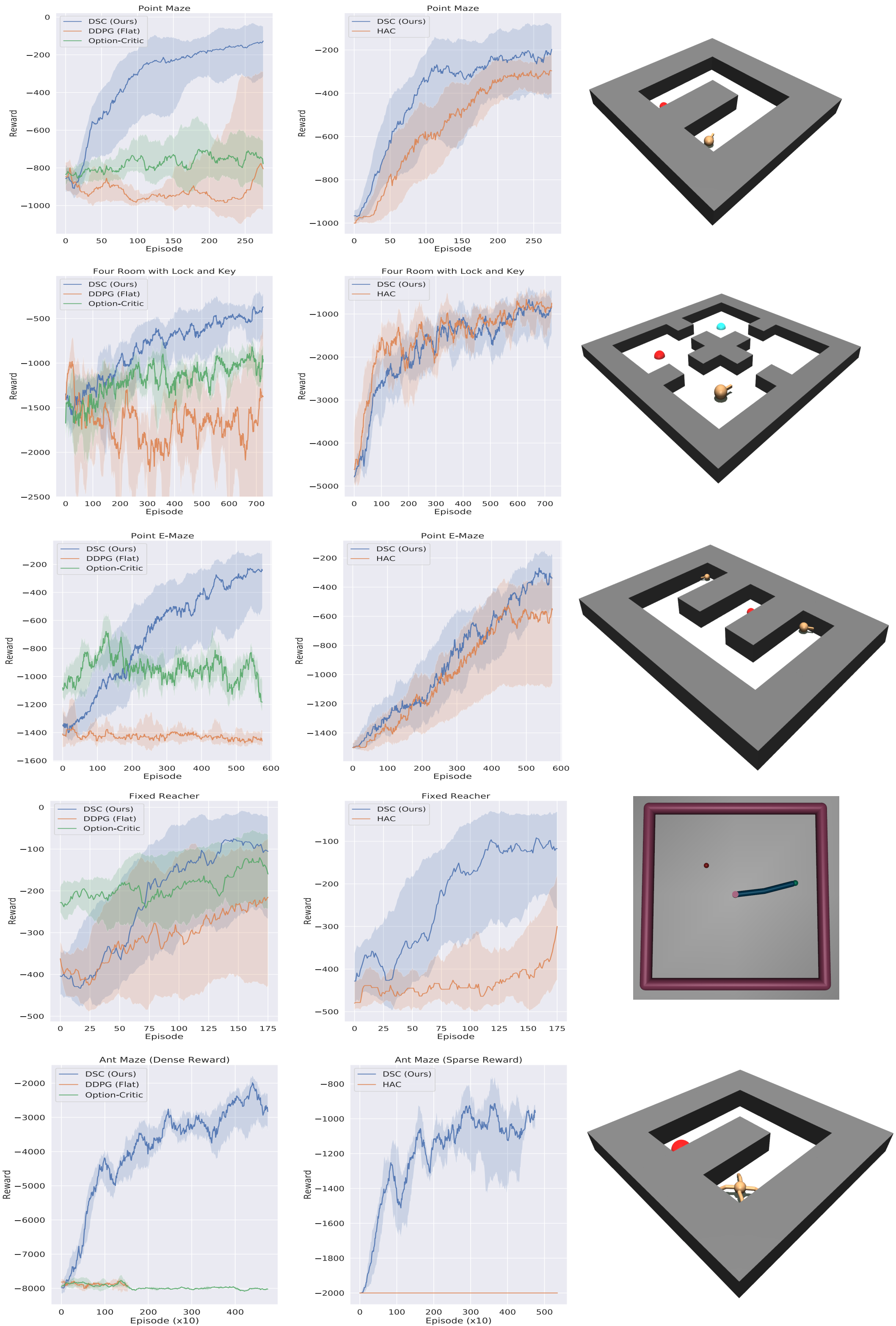
To summarize, our algorithm proceeds as follows: (1) Collect trajectories that trigger new option o_k's termination condition β_{o_k}. (2) Train o_k's option policy π_{o_k}. (3) Learn o_k's initiation set classifier $\mathcal{I}_{o_k}$. (4) Add o_k to the agent's option repertoire. (5) Create a new option o_{k+1} such that $\beta_{o_{k+1}} = \mathcal{I}_{o_k}$. (6) Train policy over options $\pi_{\mathcal{O}}$. Steps 1, 3, 4 and 5 continue until the MDP's start state is inside some option's initiation set. Continue steps 2 and 6 indefinitely.

3.2 Experiments

We test our algorithm in five tasks that exhibit a strong hierarchical structure: (1) Point-Maze (Duan et al., 2016), (2) Four Rooms with Lock and Key, (3) Reacher (Brockman et al., 2016), (4) Point E-Maze and (5) Ant-Maze (Duan et al., 2016; Brockman et al., 2016).

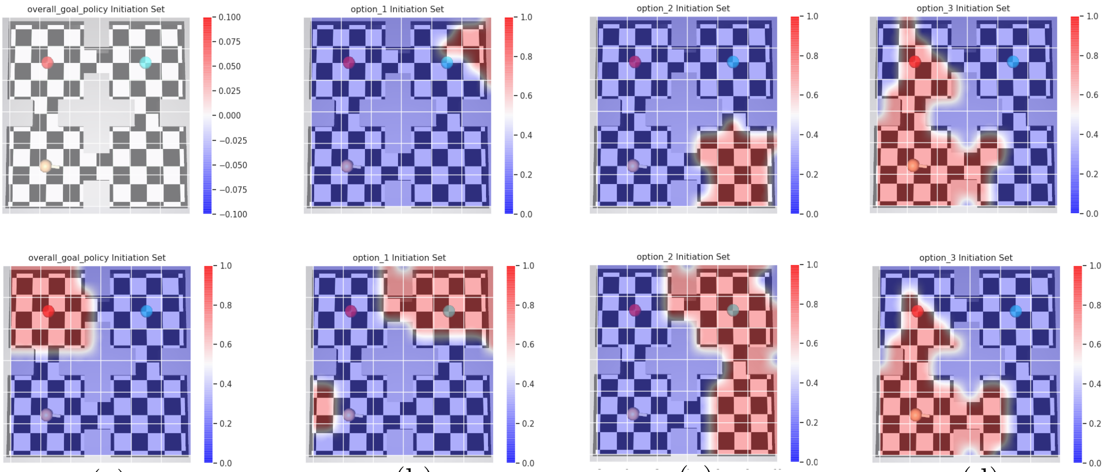
Four Rooms with Lock and Key: In this task, a point agent (Duan et al., 2016) is placed in the Four Rooms environment (Sutton et al., 1999). It must pick up the key (blue sphere in the top-right room in Figure 3.1(c), row 2) and *then* navigate to the lock (red sphere in the top-left room). The agent's state space consists of its position, orientation, linear velocity, rotational velocity and a `has_key` indicator variable. If it reaches the lock with the key in its possession, its episode terminates with a sparse reward of 0; otherwise it gets a step penalty of -1. If we wish to autonomously discover the importance of the key, (i.e, without any corresponding extrinsic rewards) a distance-based dense reward such as that used in related work (Nachum et al., 2018) would be infeasible.

Point E-Maze: This task extends the benchmark U-shaped Point-Maze task (Duan et al., 2016) so that the agent has two possible start locations - on the top and bottom rungs of the E-shaped maze respectively. We include this task to demonstrate our algorithm's ability to construct skill trees.



(a) (b) (c)

Figure 3.1: (a) Learning curves comparing deep skill chaining (DSC), a flat agent (DDPG) and Option-Critic. (b) Comparison with Hierarchical Actor Critic (HAC). (c) the continuous control tasks corresponding to the learning curves in (a) and (b). Solid lines represent median reward per episode, with error bands denoting one standard deviation. Our algorithm remains the same between (a) and (b). All curves are averaged over 20 runs, except for Ant Maze which was averaged over 5 runs.



(a) (b) (c) (d)

Figure 3.2: Initiation sets of options learned in the Lock and Key task. Blue sphere in top-right room represents the key, red sphere in top-left room represents the lock. Red regions represent states inside the initiation classifier of learned skills, whereas blue/gray regions represent states outside of it. Each column represents an option - the top row corresponding to the initiation set when `has_key` is false and the bottom row corresponding to the initiation set when `has_key` is true.

3.2.1 Comparative Analyses

We compared the performance of our algorithm to DDPG, Option-Critic and Hierarchical Actor-Critic (HAC), in the conditions most similar to those in which they were originally evaluated. For instance, in the Ant-Maze task we compare against Option-Critic under a dense-reward formulation of the problem while comparing to HAC under a sparse-reward version of the same task. As a result, we show the learning curves comparing against them on different plots (columns (a) and (b) in Figure 3.1 respectively) to emphasize the difference between the algorithms, the settings in which they are applicable, and the way they are evaluated.

Comparison with DDPG and Option-Critic: Figure 3.1(a) shows the results of comparing our proposed algorithm (DSC) with a flat RL agent (DDPG) and the version of Option-Critic designed for continuous action spaces (PPOC).[2] Deep skill chaining comfortably outperforms both baselines. Both DSC and DDPG use the same exploration

---

[2] PPOC author's implementation: https://github.com/mklissa/PPOC

strategy in which $a_t = \pi_\theta(s_t) + \eta_t$ where $\eta_t \sim N(0, \epsilon_t)$. Option-Critic, on the other hand, learns a stochastic policy $\pi_\theta(a_t|s_t)$ and thus has baked-in exploration (Sutton and Barto, 2018, Ch. 13), precluding the need for additive noise during action selection. We hypothesize that this difference in exploration strategies is the reason Option-Critic initially performs better than both DDPG and DSC in the Reacher and Point E-Maze tasks.

Comparison with Hierarchical Actor-Critic: We compare our algorithm to Hierarchical Actor-Critic (HAC) (Levy et al., 2019), which has recently outperformed other hierarchical reinforcement learning methods (Nachum et al., 2018; Vezhnevets et al., 2017) on a wide variety of tasks. [3] A noteworthy property of the HAC agent is that it may prematurely terminate its training episodes to prevent flooding its replay buffer with uninformative transitions. The length of each training episode in DSC however, is fixed and determined by the test environment. Unless the agent reaches the goal state, its episode lasts for the entirety of its episodic budget (e.g, this would be 1000 timesteps in the Point-Maze environment). Thus, to compare the two algorithms, we perform periodic test rollouts wherein all networks are frozen and both algorithms have the same time budget to solve the given task. Furthermore, since both DSC and HAC learn deterministic policies, we set $\epsilon_t = 0$ during these test rollouts. When comparing to HAC, we perform 1 test rollout after each training episode in all tasks except for Ant-Maze, where we average performance over 5 test rollouts every 10 episodes.

Figure 3.1(b) shows that DSC outperforms HAC in all environments except for Four Rooms with a Lock and Key, where their performance is similar, even though DSC does not use Hindsight Experience Replay (Andrychowicz et al., 2017) to deal with the sparse reward nature of this task.

---

[3] HAC author's implementation: https://github.com/andrew-j-levy/Hierarchical-Actor-Critc-HAC-

3.2.2 Interpreting Learned Skills

Figure 3.2 visualizes the initiation set classifiers of options discovered by DSC in Four Rooms with a Lock and Key. Despite not getting any extrinsic reward for picking up the key, DSC discovers the following skill chain: the options shown in Figure 3.2 columns (c) and (d) bring the agent to the room with the key. The option shown in column (b) then picks up the key (top row) and then takes the agent to the room with the lock (bottom row). Finally, the option in column (a) solves the overall problem by navigating to the lock with the key.

Figure 3.3 shows that DSC is able to learn options that induce simple, efficient policies along different segments of the state-space. Furthermore, it illustrates that in some states, the policy over options prefers primitive actions (shown in black) over learned skills. This suggests that DSC is robust to situations in which it constructs poor options or is unable to learn a good option policy in certain portions of the state-space. In particular, Figure 3.3 (d) shows how DSC constructs a skill tree to solve a problem with two distinct start states. It learns a common option near the goal (shown in blue), which then branches off into two different chains leading to its two different start states respectively.

3.3 Discussion and Conclusion

Deep skill chaining breaks complex long-horizon problems into a series of sub-problems and learns policies that solve those sub-problems. By doing so, it provides a significant performance boost when compared to a flat learning agent in all of the tasks considered in Section 3.2.

We show superior performance when compared to Option-Critic, the leading framework for option discovery in continuous domains. A significant drawback of Option-Critic is that it assumes that all options are executable from everywhere in the state-space. By contrast, deep skill chaining explicitly learns initiation set classifiers. As a result,

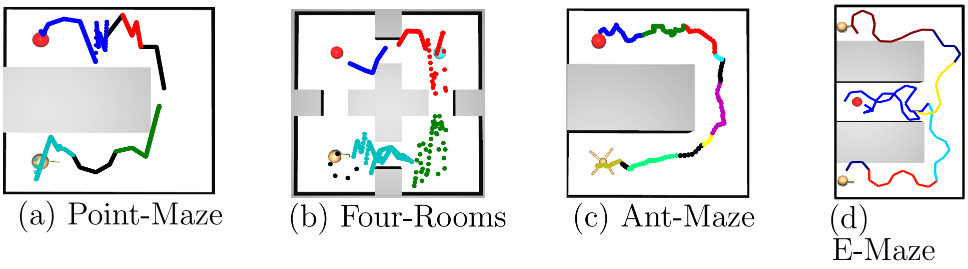

(a) Point-Maze (b) Four-Rooms (c) Ant-Maze (d) E-Maze

Figure 3.3: Solution trajectories found by deep skill chaining. Sub-figure (d) shows two trajectories corresponding to the two possible initial locations in this task. Black points denote states in which $\pi_{\mathcal{O}}$ chose primitive actions, other colors denote temporally extended option executions.

learned skills specialize in different regions of the state-space and do not have to bear the burden of learning representations for states that lie far outside of their initiation region. Furthermore, each option in the Option-Critic architecture leverages the same state-abstraction to learn option-specific value functions and policies, while deep skill chaining permits each skill to construct its own skill-specific state-abstraction (Konidaris and Barto, 2009a). An advantage of using Option-Critic over DSC is that it is not confined to goal-oriented tasks and can work in tasks which require continually maximizing non-sparse rewards.

Section 3.2 also shows that deep skill chaining outperforms HAC in four out of five domains, while achieving comparable performance in one. We note that even though HAC was designed to work in the multi-goal setting, we test it here in the more constrained single-goal setting. Consequently, we argue that in problems which permit a stationary set of target events (like the ones considered here), deep skill chaining provides a favorable alternative to HAC. Furthermore, HAC depends on Hindsight Experience Replay (HER) to train the different layers of their hierarchy. Deep skill chaining shows the benefits of using hierarchies even in the absence of such data augmentation techniques but including them should yield additional performance benefits in sparse-reward tasks.

A drawback of deep skill chaining is that, because it builds skills backward from the goal, its performance in large state-spaces is dependent on a good exploration algorithm. We used the naive exploration strategy of adding Gaussian noise to chosen actions (Lillicrap et al., 2015; Fujimoto et al., 2018) since the exploration question is orthogonal to the ideas presented here. The lack of a sophisticated exploration algorithm also explains the higher variance in performance in the Point-Maze task in Figure 3.1. Combining effective exploration (Machado et al., 2018c; Jinnai et al., 2020) with DSC's high reliability of triggering target events is a promising avenue for future work.

We presented a new skill discovery algorithm that can solve high-dimensional goal-oriented tasks far more reliably than flat RL agents and other popular hierarchical methods. To our knowledge, DSC is the first deep option discovery algorithm that does not treat the number of options as a fixed and costly hyperparameter. Furthermore, where other deep option discovery techniques have struggled to show consistent improvements over flat agents in the single task setting (Zhang and Whiteson, 2019; Smith et al., 7 15; Harb et al., 2018; Klissarov et al., 2017), we show the benefits of hierarchies for solving such challenging problems.

CHAPTER 4

Robustly Learning Composable Options in Deep Reinforcement Learning

Hierarchical approaches succeed by allowing the agent to sequentially execute high-level actions. This intuition has led to several skill-discovery algorithms that explicitly satisfy the composability objective: executing one skill takes the agent to a state where it can execute another. Such algorithms are widespread in the literature; from control-theory (Lyapunov, 1992; Burridge et al., 1999; Tedrake, 2009), to robotics (Lozano-Perez et al., 1984; Kaelbling and Lozano-Pérez, 2017), to the online (Randløv et al., 2000; Konidaris and Barto, 2009b; Shoeleh and Asadpour, 2017; Bagaria and Konidaris, 2020b) and batch RL (Singh et al., 2020) settings.

Even when skills are not explicitly constructed to be sequentially executable, they must eventually be sequenced to solve goal-directed tasks (Sharma et al., 2020c; Frans et al., 2018). Additionally, skills that can be reliably sequenced can support abstract, high-level planning (Konidaris et al., 2018).

The core difficulty that arises when discovering composable skills is that of *non-stationary subgoals*. Skill-discovery algorithms that explicitly optimize for composability must learn the regions (called *initiation sets* (Sutton et al., 1999)) from which each skill can be successfully executed. To construct sequentially executable skills, the initiation set of one skill becomes the subgoal of a new skill (Konidaris and Barto, 2009b). These initiation sets are constantly changing as the agent learns, causing the target (and therefore reward function) of successive skills to also change over time, destabilizing composability and complicating learning.

We propose three methods to combat this problem. First, we propose to stabilize learned initiation sets using a dual-classifier approach. An optimistic classifier determines when the agent can execute the skill, which encourages exploration. Meanwhile, a pessimistic classifier is used as a subgoal target, which is likely to grow but unlikely to shrink, leading to stable chains. Next, to create skill policies that are robust to non-stationary subgoals, we propose to use goal-conditioned policies (Schaul et al., 2015) for each skill. Such policies are robust to subgoal changes because they can always be re-targeted towards a state in the new subgoal region. Finally, while the majority of skill-discovery work has been model-free, we show that using model-based RL to learn skill policies leads to a substantially more robust skills.

To evaluate our proposed changes, we consider four sparse-reward continuous control problems in MuJoCo (Todorov et al., 2012). Compared to a flat model-free solution (Fujimoto et al., 2018; Andrychowicz et al., 2017), our agent achieves at least two orders of magnitude better sample efficiency. While a flat model-based solution (Nagabandi et al., 2018) is unable to solve any of the problems considered in this paper, our model-based hierarchy solves them with relative ease. Finally, our agent achieves at least a 5× improvement in sample-efficiency over a state-of-the-art skill-discovery algorithm (Bagaria and Konidaris, 2020b).

4.1 Robustly Learning Composable Options

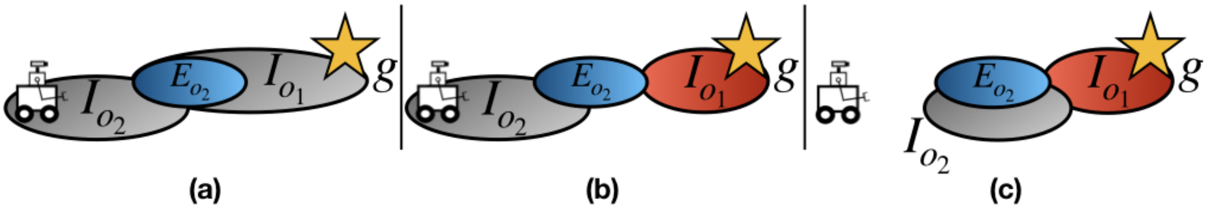


Figure 4.1: The non-stationary subgoal problem: (a) an agent has two options o_1, o_2 such that o_2 targets I_{o_1} and o_1 targets goal g; E_{o_2} is the next state distribution of π_{o_2}. (b) If I_{o_1} shrinks, executing o_2 no longer allows the agent to execute o_1 (since $E_{o_2} \not\subseteq I_{o_1}$). (c) This causes β_{o_2} to shift forward, which invalidates the previous policy π_{o_2}, which in turn causes $\mathcal{I}_{o_2}$ to shrink.

The core difficulty in learning composable options is that, during learning, all three components $(\mathcal{I}_o, \beta_o, \pi_o)$ of every option are simultaneously in flux. This difficulty is compounded by the relationship between the initiation region of one option and the subgoal region of another—changes to one option's initiation region changes another's subgoal, in turn changing its own policy and initiation region. These changes cascade to downstream options, propagating instability and causing chains of composable options to become unreliable or even break. This situation—which can happen whenever composable options are being learned simultaneously—is illustrated in Figure 4.1.

To ameliorate this difficulty, which we call *nonstationary subgoals*, we propose to robustify the learning process of all three option components. First, we propose a two-classifier representation of the initiation region that provides a stable subgoal target while encouraging exploration. Second, we use hindsight-experience replay to learn a generalized policy that enables us to target subgoal states that maximize the probability of being able to execute the successor option. Finally, to manage the difficulty of learning so many functions in parallel, we leverage the higher stability of model-based RL methods to learn more robust option policies.

4.1.1 Dual Initiation Classifiers to Avoid Shrinkage

Previous approaches learned a single binary classifier to represent $\mathcal{I}_o$ for each option o in the skill chain. If the option execution succeeds (i.e, the agent reaches β_o), it adds a new positive example for further refining $\mathcal{I}_o$. Since the positive example is from a region already inside the classifier, a successful execution leaves the classifier's decision boundary largely unchanged. Alternatively, if the option execution fails to reach β_o, the agent gets a negative example for the next training iteration of $\mathcal{I}_o$. Since this negative example comes from a region inside the classifier, it often shrinks the initiation region over time, with no opportunity to expand.

To avoid this issue, we propose a dual-classifier parameterization of $\mathcal{I}_o$—representing it using both optimistic and pessimistic classifiers. The pessimistic classifier represents the states from which we are highly confident that option execution will succeed, and so is a stable region for other options to target. However, if the agent could only ever choose to execute the option from inside this classifier, exploration would be hindered because the option would be prevented from expanding to new regions. To encourage exploration outside the pessimistic region, we also use an optimistic classifier to represent states where the agent can choose to execute the option. Eventually, the two classifiers should converge to approximate the "true" initiation region of the option.

Many techniques could be used to learn these two classifiers; we learn the optimistic classifier using a two-class SVM (Cortes and Vapnik, 1995) and the pessimistic classifier using a one-class SVM (Tax and Duin, 1999).

4.1.2 Robust Subgoals via Goal State Selection

We next turn to termination regions. The "chainability" of options o_i and o_j implies that the subgoal region β_{o_i} is a subset of the initiation region $\mathcal{I}_{o_j}$. As $\mathcal{I}_{o_j}$ is refined over time, the subgoal region β_{o_i} also changes, and consequently, so does the option's terminating reward. Since learning is highly sensitive to even small changes in the reward

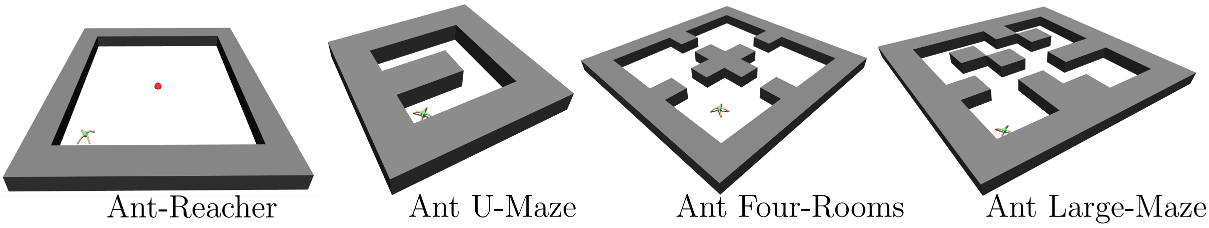


Figure 4.2: Maze navigation problems used to test our algorithm. These tasks require that the agent simultaneously learn good gait policies that stabilize the "ant" robot and navigate to a distant goal in the presence of unknown obstacles.

function (Packer et al., 2018), this creates instability in learning π_{o_i}. DSC (Bagaria and Konidaris, 2020b) mitigates this issue by freezing $\mathcal{I}_{o_j}$ after a fixed number of learning iterations, which can lead to the agent being stuck with poor estimates of $\mathcal{I}_{o_j}$ that hinder reliable skill composition.

Nevertheless, an option's subgoal region will unavoidably change continually as its target option refines its initiation classifier. To be robust to changes in β_o, we propose to make each option policy more flexible: rather than a fixed policy π_o, each option learns a goal-conditioned policy $\pi_o(s|g)$ using hindsight experience replay (HER). By conditioning the option policy π_o on goals sampled from β_o, and postponing selecting g until option execution time, the agent learns a policy robust to non-stationarity in β_o.

The goal-conditioned option policy strategy necessitates a strategy for sampling subgoal states from β_o. We propose optimizing two objectives for choosing subgoal states: (a) robustness and (b) hierarchical optimality.

Selecting Subgoal States for Robustness

Each option policy is rewarded for reaching its termination region; from its own perspective, all states in its termination region are equally rewarding. However, for a subsequent option o, some start states are better than others. How can we pick a subgoal for one option so that we increase the probability that a successive option execution will succeed?

One way is to evaluate the probability of every positive example of o succeeding (by querying the probabilistic classifier representing I_o), but that becomes computationally very expensive as the agent gathers experience. Instead, we use the simple heuristic that, over time, the agent will learn to execute the option from reliable start states, and simply sample from the set of positive examples used to train $\mathcal{I}_o$.

Selecting Subgoal States for Hierarchical Optimality

The method above results in feasible trajectories that are, at best, *recursively optimal* (Barto and Mahadevan, 2003). We would prefer to pick subgoals for each option in the skill-chain so that the overall solution trajectory is approximately *hierarchically optimal* (Barto and Mahadevan, 2003).

To select such subgoals, we first store the states $\hat{\beta}_o$ in which each option o triggered β_o. Then, we use dynamic programming (DP) to distribute the value of reaching the MDP's goal g to all the $\hat{\beta}_o$s along the skill chain. This results in a value table $\tilde{Q} : \mathcal{S} \times \mathcal{S} \to \mathbb{R}$ that can be used to pick a subgoal s_g for the current option o_t from state s_t: $s_g = \arg\max_{s \in \hat{\beta}_{o_t}} \tilde{Q}(s_t, s)$. The quality of the resulting sub-goals depends on how well $\hat{\beta}_o$ approximates β_o. If it is a perfect approximation, this DP algorithm yields the hierarchically optimal solution; otherwise, it yields a near-optimal solution.

4.1.3 Learning Robust Option Policies

Finally, we consider the third component of an option: its policy. Given the number of components simultaneously being learned in hierarchical algorithms, policy learning must be highly stable for the agent to succeed. Although model-free methods are commonplace for learning option policies, model-based methods are often more stable and sample-efficient (Deisenroth and Rasmussen, 2011).

We therefore propose learning option policies using model-based RL. We follow Nagabandi et al. (2018) to learn a dynamics model f_ξ. However, Equation their search method is insufficient for good action-selection in sparse reward problems. So, we solve the following infinite-horizon optimization problem and then execute the first action (Lowrey et al., 2019):

$$\pi(s|g) = \arg\max_{a \in \mathcal{U}^{|A|}} \sum_{t=1}^{H} \gamma^{t-1} \mathcal{R}(s_t, g) + \gamma^{H} V_\phi(s_H|g). \tag{4.1}$$

We follow the same procedure as the model-free variant of our algorithm to learn the terminal value function V_ϕ.

4.2 Experiments

Our experiments aim to answer the following questions: 1) do the proposed improvements increase the probability with which a sequence of options can be successfully composed? 2) Does the subgoal selection algorithm approximate hierarchically optimal trajectories? 3) How does the proposed algorithm compare to flat RL and other skill-discovery algorithms?

To answer these questions, we use a test-bed comprising four continuous-control maze-navigation tasks (shown in Figure 5.3) involving an "ant" robot simulated using MuJoCo (Todorov et al., 2012; Duan et al., 2016; Fu et al., 2020).

Sparse vs Dense Rewards. Dense reward functions, although commonplace in RL, are often problematic because they demand cumbersome engineering (Yu et al., 2020), can lead to sub-optimal solutions (Ng et al., 1999) or reduce the problem so much that simple search might outperform RL (Mania et al., 2018). Since hierarchies are a promising way to address the challenges of sparse rewards, we evaluate all algorithms in the sparse reward setting.

4.2.1 Implementation Details

We use TD3 (Fujimoto et al., 2018) and HER to learn goal-conditioned value functions in all our experiments. Transitions seen by all options were used to train a single dynamics model f_ξ, critic V_ϕ, and actor π_ζ. V_ϕ and π_ζ were used for selecting actions in the model-free case. Only V_ϕ from TD3 was used in the model-based case, where we approximately solved Equation 4.1 for action-selection.

4.2.2 Evaluating Robustness

We first evaluate whether the proposed changes increase the composability of discovered options, by measuring the robustness of skill-chains constructed in Ant U-Maze. We measure the robustness of a sequence of options as the probability that executing each option would take the agent to a state from which it could successfully execute the next option (until it finally reaches the goal). We call this the *chaining probability*:

$$p_{chain}(o_1, ..., o_N) = \prod_{i=1}^{N} p(s_i \sim \beta_{o_{i-1}} \in \mathcal{I}_{o_i}), \tag{4.2}$$

where $p(s_i \sim \beta_{o_{i-1}} \in \mathcal{I}_{o_i})$ represents the open-loop probability that the agent will be able to execute o_i after executing option o_{i-1}. We approximate each probability term by the empirical "success rate" of the option:

$$p(s_i \sim \beta_{o_{i-1}} \in \mathcal{I}_{o_i}) \approx \frac{\#successes(o_{i-1})}{\#executions(o_{i-1})},$$

where $\#successes(o_{i-1})$ is the number of times option $\pi_{o_{i-1}}$ was able to reach its subgoal $\beta_{o_{i-1}}$.

We ablate our proposals by comparing the robustness of the following versions of our algorithm:

1. Baseline DSC: The deep skill chaining algorithm from Bagaria and Konidaris (2020b)

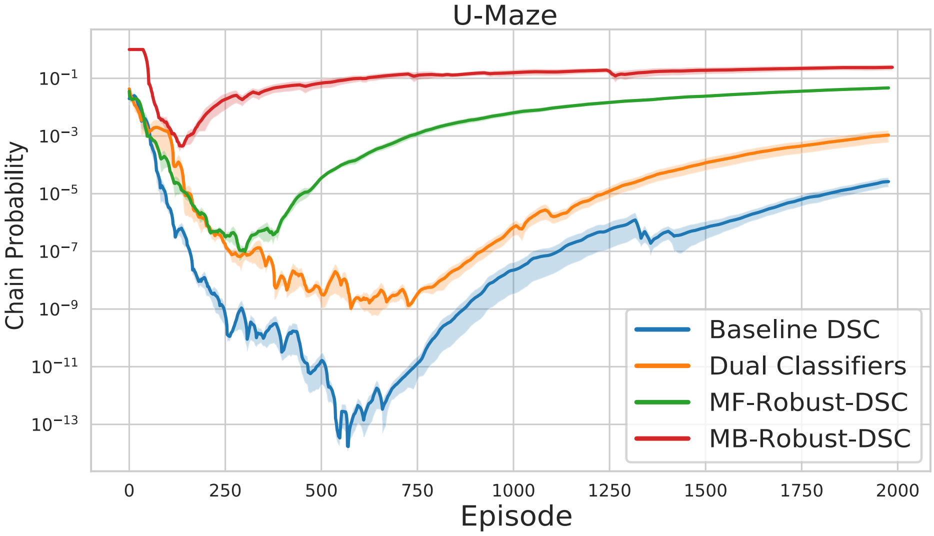


Figure 4.3: Comparing the robustness of skill chains discovered in the Ant U-Maze domain. For a description of the "chain probability" metric, please refer to Section 4.2.2. Vertical axis in log-scale; curves are averaged over 5 runs; shaded regions denote standard error.

that we aim to improve.

2. Dual Classifiers: We add the dual classifier approach from Section 4.1.1 to the baseline DSC.
3. MF-Robust-DSC: We add goal-conditioned policies to (2); this is the model-free version of the algorithm from Section 4.1.2.
4. MB-Robust-DSC: We add model-based policies to (3); this is the version of the algorithm from Section 4.1.3.

Results. Figure 4.3 shows that each of our proposals successively increases the robustness of the discovered skill-chain. The shape of the curves warrants discussion: as the agent discovers new options, its chain probability, at first, decreases. This is for two reasons: (a) more terms between 0 and 1 are included in the product of Equation 4.2 and (b) when initialized, option policies are not strong enough to reach their subgoals. Over time, two factors are responsible for the increasing trend: (a) the agent has discovered as many options as it needs to solve the problem, at which point no more terms are added to

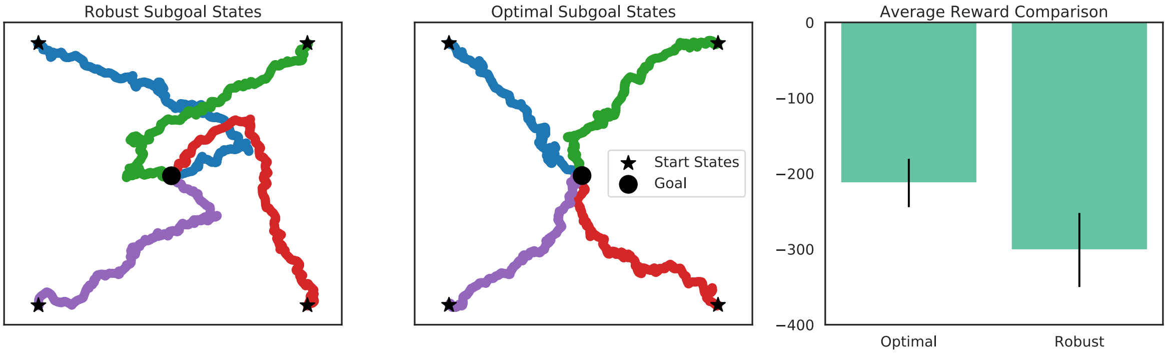


Figure 4.4: DSC trajectories (only the coordinates of the CoM are visualized) in Ant-Reacher when selecting subgoals *(left)* for robustness and *(middle)* using our dynamic programming algorithm. *(right)* When using DP, the agent scores better average reward (averaged over 5 runs; bars represent standard error; higher is better).

the product and (b) each option's policy improves. Finally, the small absolute values on the vertical axis is due to the metric being an open-loop probability—we report the closed-loop success rate of the algorithm later in Section 4.2.4.

4.2.3 Evaluating Hierarchical Optimality

To compare the two subgoal selection strategies outlined in section 4.1.2, we ran the model-based variant of our algorithm on the Ant-Reacher domain. The goal state corresponds to the center of mass (CoM) of the ant being at $(0, 0)$; the starting position of the ant is sampled uniformly from $(-10, 10)$. The learned initiation classifiers tended to lie in approximately concentric circles around the goal state (shown in Figure 4.5). After both algorithm variants were trained for 1000 episodes, they were evaluated on how many steps it would take them to reach the goal when starting in the four corners of the domain $\{(\pm 9, \pm 9)\}$.

Results. Robust subgoal selection often yielded subgoal states on the "other side" of the goal—if the robot was on the bottom-left of the domain, it could pick a subgoal closer to the top-right. This led to the sub-optimal trajectories shown in Figure 4.4a, where the ant would often over-shoot the goal. By contrast, the dynamic programming procedure picked subgoal states that were always between the agent's current state and the overall

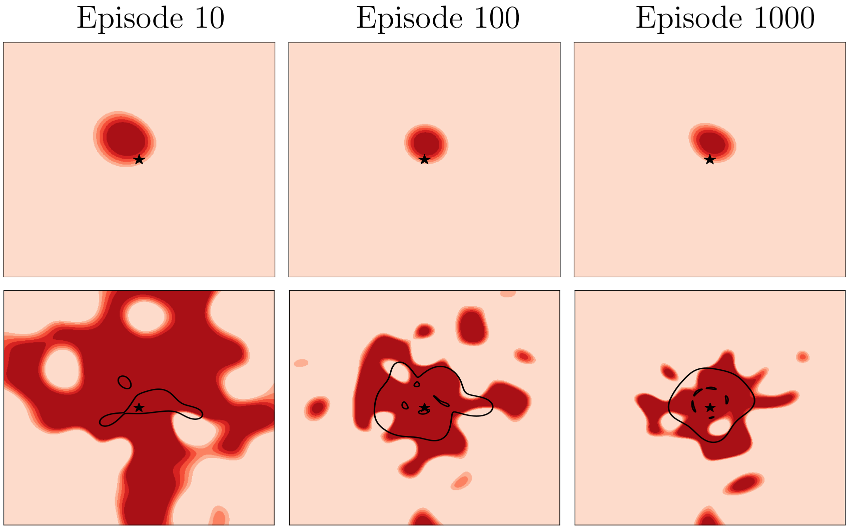


Figure 4.5: Initiation regions learned for the option that drives the agent to the goal state (visualized as a star in the center of each plot) in Ant-Reacher. The horizontal and vertical axes represent the x, y positions of the ant's CoM. The top row visualizes the classifiers learned by the baseline DSC algorithm; the bottom row corresponds to the Robust DSC algorithm. In the bottom row, the optimistic classifier is visualized in dark red, the pessimistic classifier as the black contour lines. The Robust DSC agent learns a bigger initiation region, which implies larger goal regions for successive options, thereby making their policy learning process easier.

goal—leading to much more sensible, and approximately hierarchically optimal, solution trajectories (Figure 4.4b). Figure 4.4c shows that the agent using our DP algorithm for picking subgoals collected higher average rewards, further suggesting smoother overall trajectories to the goal.

4.2.4 Learning Curves

Our final experiment compares the following algorithms on ant U-maze, large-maze and four-rooms:

1. Flat model-free baseline: we used TD3+HER (Fujimoto et al., 2018; Andrychowicz et al., 2017) as a flat model-free baseline, since it is a state-of-the-art method for continuous control (we used the same algorithm to learn our model-free option policies).
2. Flat finite-horizon model-based baseline: we used the model-based algorithm from

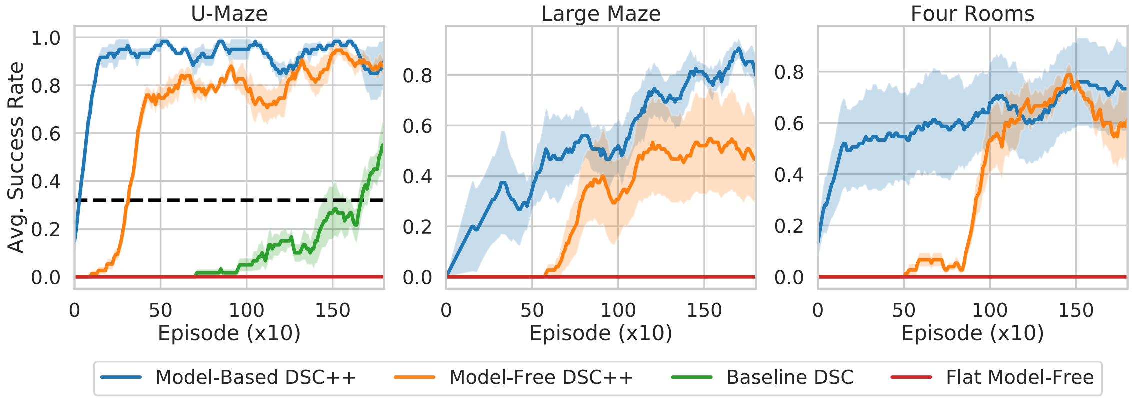


Figure 4.6: Average success rate in sparse-reward ant-navigation problems. Shaded regions denote standard error (averaged over 5 runs). The black dashed line in the leftmost subplot is the success rate of the flat model-free baseline after 12,000 episodes of training. In the middle and rightmost subplots, the green line is hidden below the red one.

Nagabandi et al. (2018) as our flat model-based baseline (we used the same algorithm to learn our model-based option policies).

3. Flat infinite-horizon model-based baseline: given that our problems are sparse-reward, we augment the model-based baseline with TD3, which is used to learn a value function that informs action-selection (Equation 4.1).

4. Deep skill chaining (DSC): DSC is our HRL baseline, because we extend DSC and it outperformed other skill-discovery methods (Bagaria and Konidaris, 2020b).

5. Model-Free DSC++ (ours): the model-free variant of our algorithm described in Section 4.1.2.

6. Model-based DSC++ (ours): the model-based variant of our algorithm described in Section 4.1.3.

We report the "average success rate" metric from Andrychowicz et al. (2017). Every 10 episodes, we ran the algorithm from a fixed start position $(0, 0)$, checking if it could reach the goal within 1000 steps. Due to its simplicity, we use the robust subgoal selection algorithm from Section 4.1.3 for rolling out our goal-conditioned option policies.

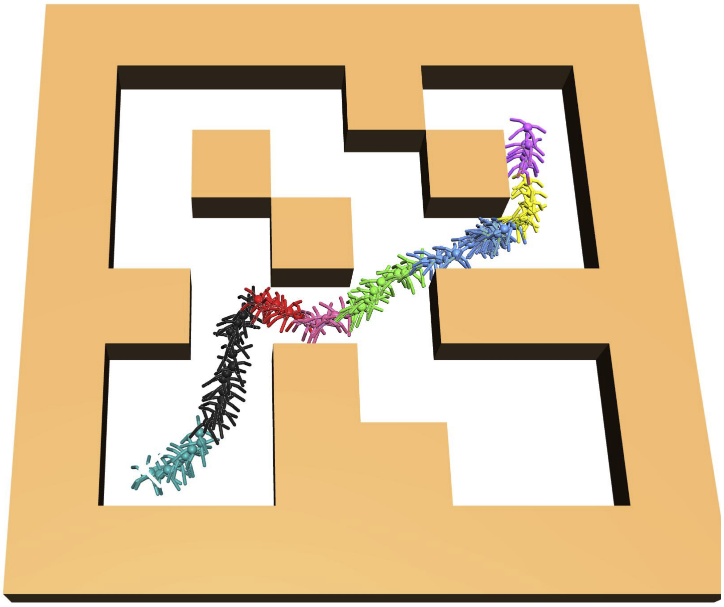


Figure 4.7: A learned solution to a long-horizon, sparse reward task that requires executing approximately 665 low-level steps (different colors denote discovered skills).

Results. Figure 4.6 shows the average success rate of competing methods averaged over 5 random seeds. Both versions of the flat model-based baseline were unable to achieve a $> 0\%$ success rate; so we leave them out of the learning curves in Figure 4.6. Ant U-Maze was the only sparse-reward problem that TD3 and the baseline DSC were able to achieve a $> 0\%$ (but TD3 had to be trained for far more episodes). The proposed algorithm (shown with blue and orange curves) were easily able to outperform all baselines including DSC. A successful trajectory is visualized in Figure 4.7.

4.3 Conclusion

The success of skill discovery in goal-directed tasks depends on learning options that can be sequentially composed. However, robust composition is challenging because of non-stationary subgoals. We proposed methods that address all three components of option learning—initiation function, termination function, and policies—to learn reliably composable options. We showed that our augmentations measurably improve the robustness of discovered skills and eventually allow us to solve more challenging problems.

CHAPTER 5

Effectively Learning Initiation Sets in Hierarchical Reinforcement Learning

General purpose agents must possess many such options to solve a wide variety of tasks (White, 2017), but increasing the size of the action set raises both the complexity of learning and the branching factor of search (Littman et al., 1995). This necessitates a mechanism for locally pruning irrelevant options: most options do not apply in a given state and the agent can lower its effective branching factor by ignoring them (Precup et al., 1998; Wen et al., 2020). Fortunately, the options framework includes the concept of *initiation sets* which captures the idea of local applicability: an option can only be executed from states inside the initiation set.

It is natural to view an initiation set as the set of states from which option execution is likely to succeed; in that case, learning it is straightforward when the option's policy is *fixed*: run the option from different states, record the outcome, and learn a classifier that predicts whether option execution will succeed at a given state (Konidaris and Barto, 2009b; Kaelbling and Lozano-Pérez, 2017). However, this is complicated by non-stationarity during learning: as the option policy changes, previously recorded outcome labels become invalid. How should the initiation classifier change in response? Another challenge is that initiation set learning suffers from a *pessimistic bias*: the option policy mostly improves for states *inside* the initiation set because option execution only occurs from such states. Therefore, if a state is excluded from the initiation set early during learning, it will likely remain excluded even as the policy improves over time. This pessimistic bias causes initiation sets to mostly shrink and rarely expand (Bagaria et al., 2021a), even in the face of policy improvement; this is potentially catastrophic for learning since the agent prematurely gives up on options in favor of primitive actions (Harb et al., 2018).

To address the issue of non-stationarity, we introduce the Initiation Value Function (IVF), a general value function (GVF) (Sutton et al., 2011; White, 2015) that predicts the probability that option execution will succeed from a given state. Since the IVF is purely predictive, it can be learned using tools from off-policy policy evaluation (OPE) (Voloshin et al., 2021) and does not interfere with the option's main task of maximizing its internal pseudo-reward function. Unlike the classification approach described above, the IVF adapts to a changing policy: as the policy improves over time, so does the IVF's estimates of its probability of success. We show how the IVF can be used as a direct estimate of the option's initiation set or as input to a *weighted* classifier that accounts for changes in the option policy.

To address the pessimistic bias of learning initiation sets, we expand the criterion for including a state in the option's initiation set: in addition to states from which option execution is likely to succeed, we also include states for which the option policy is most likely to *improve*. By identifying and adding these states to the initiation set, we mitigate the pessimistic bias and prevent initiation sets from collapsing as the option is learned.

For evaluation, we first measure how accurately we can deduce an option's initiation set in MiniGrid-FourRooms and the first screen of Montezuma's Revenge. Then, we demonstrate that our proposed methods can effectively identify promising grasps in challenging robot manipulation problems in Robosuite. Finally, by integrating our method for learning initiation sets into an existing option discovery algorithm, we solve a maze navigation problem in MuJoCo that baseline agents are unable to.

5.1 Effectively Learning Initiation Sets

The problem of learning initiation sets can be naturally framed as training a classifier where states on trajectories that achieve the subgoal are labeled as positive examples, and states on trajectories that fail to achieve the subgoal are labeled as negative examples. A probabilistic classifier trained on this data will predict the probability that an option will succeed from a given state, which is exactly the desired semantics of an initiation set. However, when initiation sets and policies have to be learned together, such a classifier is no longer a good estimator, for three reasons:

Data non-stationarity. The first reason is data non-stationarity, which refers to the phenomenon that previously collected training examples become invalid during the course of learning, but continue to impact classifier training. Consider a trajectory τ_0 obtained by rolling out an option policy $\pi_o^{t_0}$ at time t_0. If the policy is in the early stages of training, $\pi_o^{t_0}$ will likely fail to reach the subgoal region β_o and τ_0 will be labeled as a negative example ($Y(s) = 0, \forall s \in \tau_0$). At some future time t_1, the policy might improve, but the states along τ_0 will continue to have a negative label, which means that they will continue to be outside the initiation set. As a result, we will fail to capture the growing competence/reachability of $\pi_o^{t>t_0}$. A naïve solution to this problem would be to discard old training examples, but not only would that be sample-inefficient, it is also unclear at what point a training example is "old enough" to be discarded.

Pessimistic bias. The second reason, pessimism, refers to the fact that once states are deemed to be outside the initiation set, they are unlikely to switch labels because the option has low probability of being executed from them, and so policy improvement from these states is unlikely (Figure 5.1 *(right)*).

Temporal Structure. The third reason is that classification (or equivalently, Monte

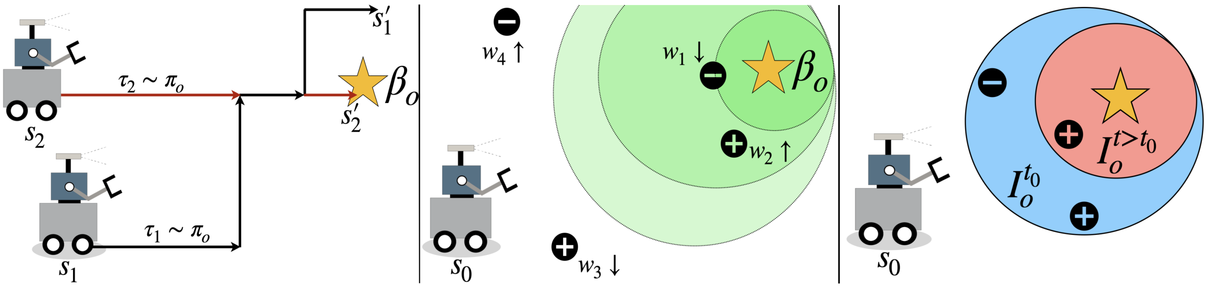


Figure 5.1: Learning an initiation set for an option targeting subgoal β_o (star). *(left)* Temporal structure in initiation set learning: even though τ_1 (black) fails to reach the goal, states along it should get credit because of the portion shared with the successful trajectory τ_2 (red); this does not happen using a simple classification approach. *(middle)* the initiation value function (IVF) $\mathcal{V}_o$ (Sec 5.1.1) is shown in green: darker the shade, higher the value; classification examples should be re-weighted to account for a changing policy. *(right)* Pessimistic bias causes initiation sets to shrink (blue → red) over time.

Carlo learning (Sutton and Barto, 2018)) does not exploit the temporal structure in the initiation set learning problem. Figure 5.1 *(left)* shows two trajectories sampled from the same option's policy: τ_1 reaches the goal and τ_2 does not. Any form of Monte Carlo estimation using this data would assign an initiation probability of 0 to most states in trajectory τ_1. However, this does not assign enough credit to most of the good decisions made along τ_1—the only way to achieve that would be via *temporal bootstrapping*. This form of generalization is not possible when using classification.

To address non-stationarity and to exploit temporal structure, we formulate a new general value function, the *Initiation Value Function* (IVF; Section 5.1.1), learned using temporal difference (TD) methods (Sutton, 1988). To address pessimism, we augment the initiation set with states from which the option policy is most likely to *improve* (Section 5.1.3).

5.1.1 Initiation Value Function (IVF)

An option o's initiation set $\mathcal{I}_o$ is the set of states from which the option policy π_o can succeed with high probability: $\mathcal{I}_o = \{s : \mathbb{P}(s' \in \beta_o | S = s, A = o) > T\}$ (T is a predefined threshold). Treating the success condition of the option as a cumulant $c_o : s \to \{0, 1\}$ and setting the timescale $\gamma_{c_o} := 1$, the corresponding GVF:

$$\mathcal{V}^{\pi_o}(s_t) = \mathbb{E}_{\pi_o}\Big[\sum_{t=0}^{t=H_o} c_o(s_{t+1}) | s_{t+1} \sim \mathcal{T}(s_t, \pi_o(s_t))\Big] = \mathbb{P}(c_o = 1 | S = s_t, O = o)$$

represents the initiation probability at state s_t (H_o is the option horizon). Note that $\mathcal{V}^{\pi_o}$ is different from V_o, the value function used by the option policy for control; this is because π_o maximizes an arbitrary reward function R_o and so the resulting value function V_o, which approximates the value of the optimal option policy $V^{\pi_o^*}$, cannot be interpreted as an initiation probability.

Using the IVF as an initiation set. To directly use the IVF as the initiation set, we need to pick a threshold T above which a state is deemed to be in the option's initiation set. Previous work (Kumar et al., 2018) reused the option value function V_o as the IVF, but had to develop heuristic schemes to pick a threshold. But since $\mathcal{V}_\phi^{\pi_o}$ outputs an interpretable number between 0 and 1 (we use a sigmoid layer at the end of ϕ to enforce this), it is easy to threshold, regardless of R_o.

Learning the IVF. Computing the probability of success of a given policy π_o^t is equivalent to the problem of off-policy policy evaluation. Our approach to policy evaluation *must* be sample-efficient—if it is not, the option policy will change before we are able to sufficiently evaluate it (Sutton et al., 2007). Furthermore, the initiation cumulant c_o is a sparse binary function, which makes learning with the Monte Carlo estimator high variance and sample inefficient. This once again underscores the importance of exploiting temporal structure of the problem using TD-learning (we use TD(0)) to estimate $\mathcal{V}_\phi^{\pi_o}$, which unlike a classifier, is able to propagate value from partial trajectories (as illustrated in Figure 5.1 *(left)*). In this view, using a classifier is equivalent to using a Monte Carlo estimate of the IVF: high-variance, sample inefficient, and unable to bootstrap.

5.1.2 Combining Classification and the Initiation Value Function

In Section 5.1.1 we connected the initiation set to a GVF, which allows us to use all the tools of value-based RL (Sutton and Barto, 2018) to learn the initiation set. However, the classification approach mentioned earlier has some attractive qualities: specifically, classification is easier than regression (Bishop and Nasrabadi, 2006) and the supervised learning community has developed powerful tools for learning classifiers using neural networks and cross-entropy loss (Goodfellow et al., 2016). To get the best of both worlds, we propose an additional method that combines value estimation and classification.

Recall that binary classification can be described as the process of finding parameters θ that minimize the cross-entropy loss $\mathcal{L}(\theta)$ in the training data $\mathcal{D} = \{(s_i, Y_i)\}_{i=0}^{|\mathcal{D}|}$ where s_i are the inputs (states in our case) and Y_i are the classification labels (whether option execution was successful from s). To deal with non-stationarity, we instead use the *weighted* binary cross-entropy loss $\mathcal{L}_w(\theta)$ which minimizes loss over a dataset $\mathcal{D} = \{s_i, Y_i, w_t(s_i)\}$ where $w_t(s_i)$, a state-dependent weight, represents the desired contribution of the training example to the classification loss (Natarajan et al., 2013).

How should we set the weights $w_t(s)$ of a training example (s_i, Y_i) in a way that reflects the evolving competence of the option policy? Since the IVF evaluates the success probability of the current option policy π_o^t, we can heuristically use it as a reflection of our certainty in the training example (s_i, Y_i)—positively (negatively) labeled states contribute more to $\mathcal{L}_w$ when the IVF prediction is high (low). Specifically, the weighting function is defined via the following simple equation:

$$w_t(s) = Y_s \mathcal{V}(s) + (1 - Y_s)(1 - \mathcal{V}(s)). \tag{5.1}$$

As the IVF estimates that the option policy is becoming more competent at state s, the classifier's uncertainty about a negative example at s will increase, thereby causing that negative example to contribute less to the classifier's decision boundary. Similarly, if the policy degrades at s, then the contribution of a positive label at s to $\mathcal{L}_w$ will go down. By weighing training examples in this way, a binary classifier is able to adapt to data non-stationarity that results from a changing option policy.

At the end of every option execution, we therefore recompute weights using Equation 5.1 and re-train option initiation classifiers using a weighted cross-entropy loss $\mathcal{L}_w$.

Policy Improvement Prior. We can use the fact that policies in RL usually improve over time to guide classifier training. To do this, we set the weight of positive examples $w_t(s)$ as 1 throughout training. The idea is that a state which leads to success once is likely to do so again with an even better policy. The weights of negative examples are allowed to vary with the IVF as in Eq 5.1.

5.1.3 Overcoming Pessimistic Bias

Unsuccessful option trajectories cause the initiation set to shrink. When a state is outside the initiation set, the option can no longer be executed from there. So even if the option could succeed from that state in the future, it remains outside the initiation set and the option ends up with a smaller region of competence than it needs to. This issue, which we call the pessimistic bias of learning initiation sets, can prevent the learning of useful options and lowers the effectiveness of hierarchical RL agents[1].

To mitigate this pessimistic bias, we expand the initiation set to also include states from which policy improvement is most likely:

$$\mathcal{I}_o = \{s : \mathcal{V}_o(s) + \mathcal{B}(s) > T, s \in \mathcal{S}\}.$$

Effectively identifying such states in high-dimensional spaces is the topic of bonus-based exploration in deep RL (Taïga et al., 2019). We propose two simple approaches:

---

[1] This issue is analogous to non-stationary dynamics in flat MDPs; see Section 5.1.4.

1. **Competence progress** attempts to capture regions where a policy is either improving or regressing (Şimşek and Barto, 2006; Stout and Barto, 2010; Baranes and Oudeyer, 2013). This can be computed as changes in the IVF over time: $\mathcal{B}_1(s) = \left|\mathcal{V}_o^t(s) - \mathcal{V}_o^{t-K}(s)\right|$, where $\mathcal{V}_o^t$ is the current IVF and $\mathcal{V}_o^{t-K}$ is the IVF estimate K timesteps ago (obtained using the target network).
2. **Count-based bonus** approach keeps track of the number of times $N(s, o)$ option o has been executed from state s. This is then converted into an uncertainty measure: $\mathcal{B}_2(s) = c/\sqrt{N(s, o)}$, where c is a scalar hyperparameter (Strehl and Littman, 2008).

We use count-based bonuses in problems where tabular counts are readily available; otherwise, we use competence progress.

5.1.4 Pessimistic Bias: Connections to Non-Hierarchical MDPs

MDPs with actions that have non-stationary effects mirror hierarchical MDPs in which the option policies are being learned online. Figure 5.2 shows such an MDP—there are three states: s_1, s_2, s_3, two actions a and o_t, and discount factor $\gamma < 1$. Action a takes the agent one state to the right, state s_3 yields a terminating reward of 1, reseting the state to s_1. Action o_t has a non-stationary effect: it either leaves the agent at state s_1, or takes the agent to s_3. The probability with which o_t takes the agent to s_3 evolves over

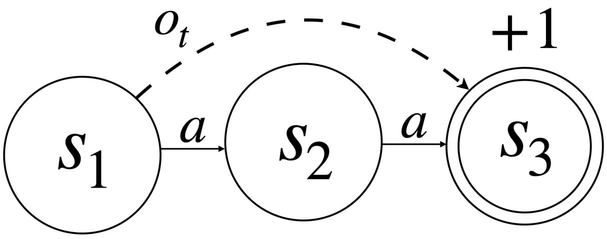


Figure 5.2: Illustration of the pessimistic bias in a tabular MDP. Action o_t has non-stationary effects, just like an option in HRL. As o_t executed more, the probability of landing in s_3 increases, otherwise it leaves the agent in s_1.

time—the more it is executed, the more consistently it takes the agent to s_3, similar to an option in HRL whose policy improves with execution.

A Q-learning agent quickly learns $Q(s_1, a) = \gamma$ and $Q(s_1, o_t) < \gamma$ leading to greedy policy $\pi(s_1) = a$. But if the agent executes o_t enough times, it will get higher return $Q(s_1, o_t) = 1 > Q(s_1, a) = \gamma$ leading to the greedy policy $\pi(s_1) = o_t$. Vanilla Q-learning alone would not discover the higher value solution because it does not account for non-stationary effect of o_t; some form of exploration is needed for that.

5.2 Experiments

We aim to evaluate whether our methods can improve hierarchical RL. First, we evaluate whether they result in better, more efficiently learned initiation sets. Second, we test if better initiation set learning improves option learning as a whole. Finally, we incorporate our changes into a state-of-the-art skill discovery method and check if resulting agent is able to outperform the baseline in a sparse-reward continuous control problem.

Implementation Details. Option policies are learned using Rainbow (Hessel et al., 2018) when the action-space is discrete and TD3 (Fujimoto et al., 2018) when it is continuous. Following Bagaria et al. (2021a), all options share the same UVFA (Schaul et al., 2015) but condition it using their own subgoals. The IVF is learned using Fitted Q-Evaluation (Le et al., 6 15), prioritized experience replay (Schaul et al., 2016) and target networks (Mnih et al., 2015). The IVF Q-function and initiation classifier are parameterized using neural networks that have the same architecture as the Rainbow/TD3. Each option has a "gestation period" of 5 (Konidaris and Barto, 2009b), which means that before the option sees 5 successful trajectories, its initiation set is optimistically initialized to be true everywhere. Since the training data for classification can be severely imbalanced, we use the standard technique of upweighting the minority class (Japkowicz and Stephen, 2002).

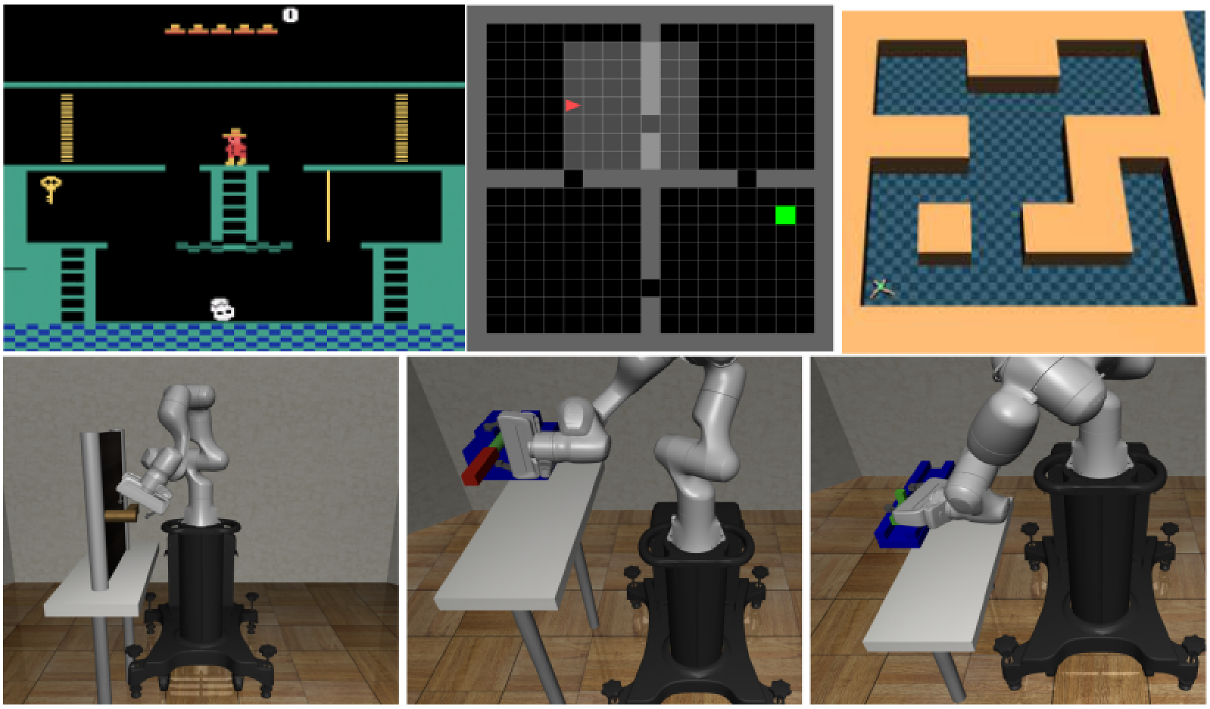


Figure 5.3: Domains used for our experiments in Section 8.5. Enumerating left to right from the top-left: first screen of Montezuma's Revenge, MiniGrid FourRooms, Ant Medium Maze, Robosuite Door, Robosuite Lever, and Robosuite Slide.

5.2.1 Measuring the Quality of Initiation Sets

Before comparing different methods based on task performance, we specifically test the quality of initiation sets learned in MINIGRID-FOURROOMS (Chevalier-Boisvert et al., 2023) and the first screen of MONTEZUMA'S REVENGE (Bellemare et al., 2013). In both domains, observations are 84 × 84 images and the action-space is discrete. For this experiment, we design start states $\mathcal{S}_0$ in each domain—in MINIGRID-FOURROOMS, $\mathcal{S}_0$ is the set of all states (since this is a small tabular domain) and in MONTEZUMA'S REVENGE, $\mathcal{S}_0$ is a series of 100 states scattered across the first screen. For the purpose of evaluating our initiation set learning algorithms, at every episode, we sweep through states $s \in \mathcal{S}_0$ and reset the simulator to s. Option termination conditions are also hand-defined: in FourRooms, we create options that target the center of each room; in Montezuma's Revenge, we define five options: those that navigate the player to each of the two doors, one that navigates the player to the bottom-right and one to the bottom-left of the first screen and finally an option that attempts to get the key.

Initiation Set Accuracy. At each state $s \in \mathcal{S}_0$, we record the initiation decision made by the learning algorithm as $\hat{\mathcal{I}}_o(s;\theta) \in \{0,1\}$. We then execute the option policy π_o from that state and record whether or not the agent reached the option's subgoal as $Y_s \in \{0,1\}$. Accuracy at state s, for option o is then given by $1(\hat{\mathcal{I}}_o(s;\theta) = Y_s)$. This process is repeated several times for all options $o \in \mathcal{O}$ and start states $s \in \mathcal{S}_0$. To be faithful to how options are learned in the online RL setting, we only use the trajectory obtained by running the option for updating the initiation classifier and the option policy if the learned classifier returned true at the start state, i.e, $\hat{\mathcal{I}}_o(s;\theta) = 1$ (in which case π_o is updated using Rainbow Hessel et al. 2018).

Initiation Set Size. Although accuracy is a natural evaluation metric, it does not fully capture the quality of the learned initiation set. For example, if the predicted initiation function returns false everywhere (i.e, if $\mathcal{I}_o(s;\theta) = 0, \forall s \in \mathcal{S}_0$), then the option policy would not get new data to improve. If the policy never improves, its true initiation set could also collapse; while such an initiation learner would register high accuracy, the option would not be useful. As a result, we additionally measure the normalized size of the "true" initiation set $|Y_s|$: this is the fraction of start states $\mathcal{S}_0$ from which Monte Carlo rollouts of the policy succeeds. A well-learned initiation set is not only accurate, it is also as large as possible,[2] reflecting a large region of option policy competence.

Competing methods. We compare the accuracy and size of the initiation sets learned by:

- *Baseline binary.* Binary classifier used to learn the initiation set. This is what is used in essentially all prior work and is the only baseline method in this experiment.
- *IVF.* Threshold the Initiation Value Function as discussed in Section 5.1.1.

---

[2] This is similar to the considerations of precision/recall in supervised learning.

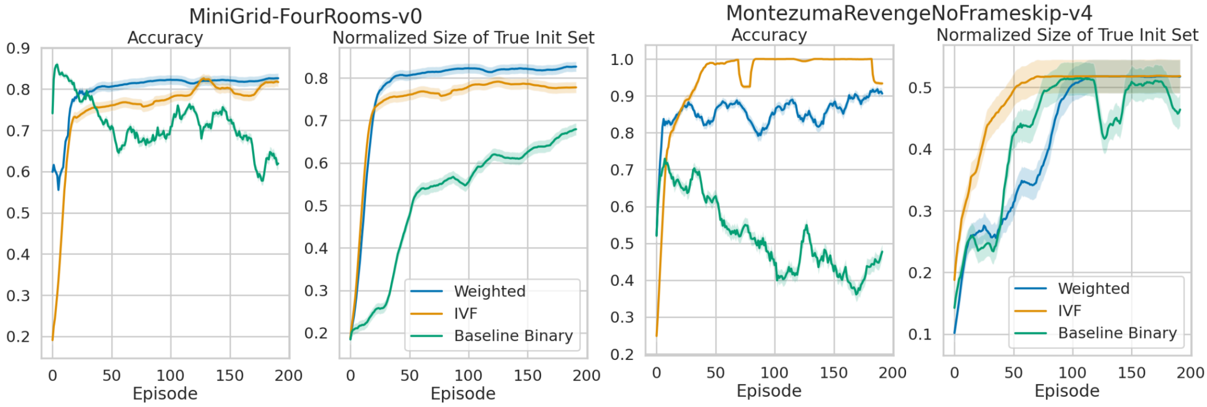


Figure 5.4: Measuring the quality of initiation sets in MiniGrid-FourRooms (left) and the first screen of Montezuma's Revenge (right). Solid lines denote mean accuracy and initiation set size, shaded regions denote standard error. All curves are averaged over all state-option pairs and 5 random seeds.

- *Weighted.* This is a weighted binary classifier discussed in Section 5.1.2.

Both the *IVF* and *Weighted* approaches use the competence progress exploration bonus described in Section 5.1.3.

Discussion of results. Figure 5.4 shows that our proposed methods significantly outperform the baseline binary classification approach, which learns classifiers that are less accurate and smaller than those learned using other techniques. Furthermore, the accuracy of the baseline binary classifier *falls* over time; this is because even though the policy improves (and hence the size of the true initiation set increases), the learned initiation set remains small and unable to adapt to an improving policy. By modifying the binary classifier using the weighting scheme described in Section 5.1.2, we are able to learn more accurate initiation sets that reflect highly competent option policies. Interestingly, the optimistic version of the IVF is necessary to learn good initiation sets in Montezuma's Revenge, but it underperforms compared to the plain IVF in FourRooms; we hypothesize that this is because FourRooms is a small and simple enough domain that an exploration bonus is not strictly necessary.

5.2.2 Robot Manipulation: Identifying Promising Grasps

In the previous section, we showed that our methods can lead to higher quality initiation sets; now we evaluate whether that can in turn improve option learning as a whole. We adopt robot manipulation as a testbed because it is a natural problem setting for evaluating initiation set learning algorithms. For example, consider a robot manipulating a hammer: not only should it choose a feasible, stable grasp on the hammer, but it should also choose one that would allow it to subsequently drive a nail. This problem, referred to as *task-specific grasping* in robotics, and has been studied extensively for a fixed policy (Kokic et al., 2020; Fang et al., 2020; Zhao et al., 2021; Wen et al., 2022; Schiavi et al., 2022); here we use RL to deduce which grasps are most likely to afford success while simultaneously learning the skill policies. Furthermore, dense reward functions are often necessary to train RL policies in robot manipulation domains (Gu et al., 2017); this section highlights that initiation probabilities can be learned even when the reward function used to train π_o is not sparse like the initiation cumulant.

We use three constrained manipulation tasks in ROBOSUITE (Zhu et al., 2020): opening a door, flipping a lever, and manipulating a sliding mechanism. Each task is modeled as a single option and the policy and initiation set are learned simultaneously. Each environment has 250 possible start states $\mathcal{S}_0$. At the start of each episode, a configuration is sampled from the initiation set learner:

$$s_0 \sim \frac{\mathcal{I}_o(s)}{\sum_{s' \in \mathcal{S}_0} \mathcal{I}_o(s')}, \forall s \in \mathcal{S}_0;$$

the manipulator is then reset to s_0. The continuous observation space is 52-dimensional, the continuous action space is 13-dimensional; we use TD3 for policy learning (Fujimoto et al., 2018).

Our algorithms are compared against two baselines: *Random*, which selects an arm

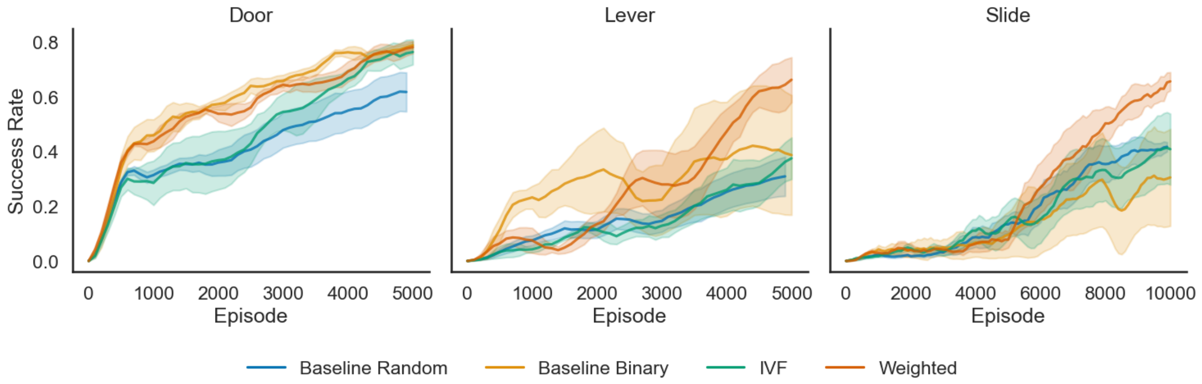


Figure 5.5: Task success rate for manipulation domains aggregated over 4 random seeds.

configuration at random at the start of each episode[3], and *Binary* which treats initiation learning as vanilla binary classification. Our algorithms, *IVF* and *Weighted*, employ the optimistic bias discussed in Section 5.1.3 by adding a count-based bonus to the predicted initiation probability.

Task success rates are shown in Figure 5.5. The DOOR task is the simplest: most of the grasp candidates afford task success. As a result, the *Random* and *Binary* baselines are competitive with our algorithms. The LEVER and SLIDE tasks are more challenging as the number of promising grasps is significantly lower. In these domains, only the *Weighted* variant is able to consistently recover a policy with success rate above 50%.

Figure 5.6 shows that our methods can identify good grasp poses without task-specific engineering or human demonstrations. This is impressive given that human data (Mandikal and Grauman, 2021, 2022) or heuristics like image segmentation (Kokic et al., 2017; Rosen et al., 2022) are typically required to learn these affordances efficiently, even in the case of a fixed policy.

5.2.3 Improving Option Discovery

Finally, to evaluate whether better option learning can improve option discovery, we integrate our approach into an existing state-of-the-art algorithm: deep skill chaining

---

[3] This baseline is equivalent to an initiation set that is uniformly true everywhere.

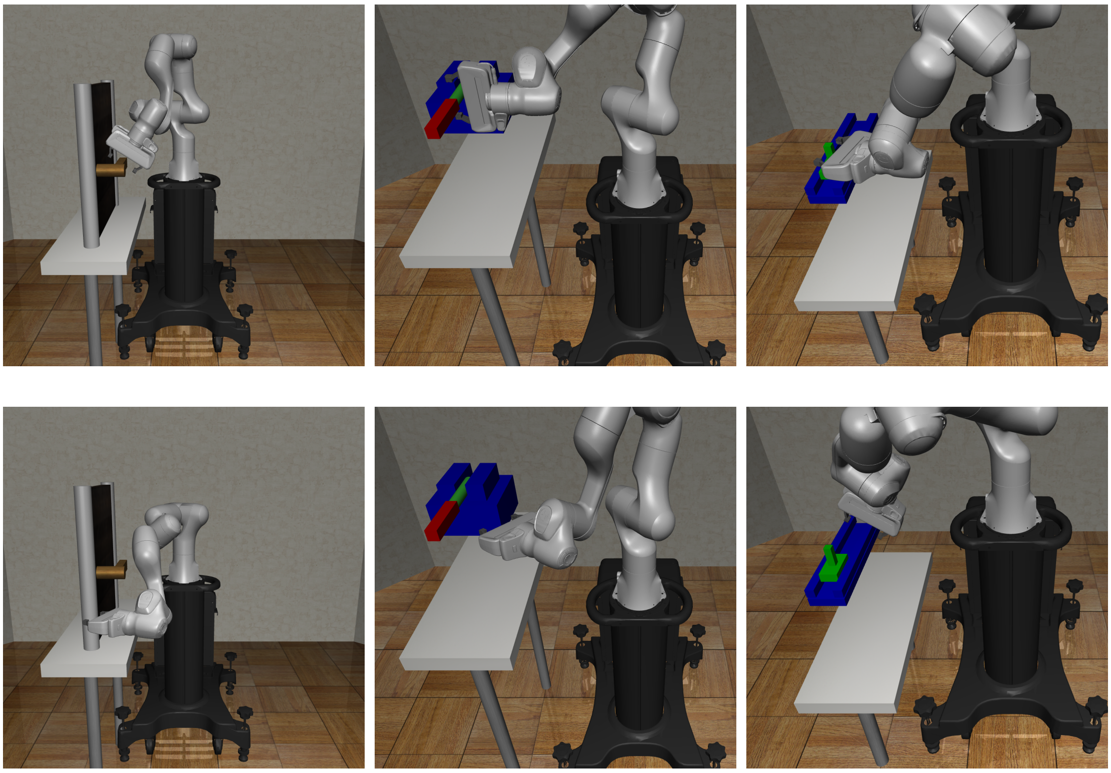


Figure 5.6: Examples of promising grasp poses *(top row)* in Door *(left)*, Lever *(middle)* and Slide *(right)*: these are high probability samples from our initiation function; contrasted with bad grasp poses *(bottom row)*, which are low probability samples and are ruled out by the initiation function.

(DSC) (Chapter 3). DSC learns a collection of options so that the subgoal region β_{o_i} of an option o_i is the initiation region $\mathcal{I}_{o_j}$ of another option o_j; by learning options that funnel into each other's initiation sets, DSC learns how to sequentially compose options to reliably achieve a goal. It does so by first learning an option that reaches the task goal, then another option that targets the first option's initiation set and so on, until the start-state is inside the initiation set of some option; for more details, please refer Chapter 3. We chose to integrate our techniques with DSC because of its focus on learning initiation sets jointly with options in an online and incremental reinforcement learning setting and because the quality of initiation sets is crucial to its performance and stability (Bagaria et al., 2021a).

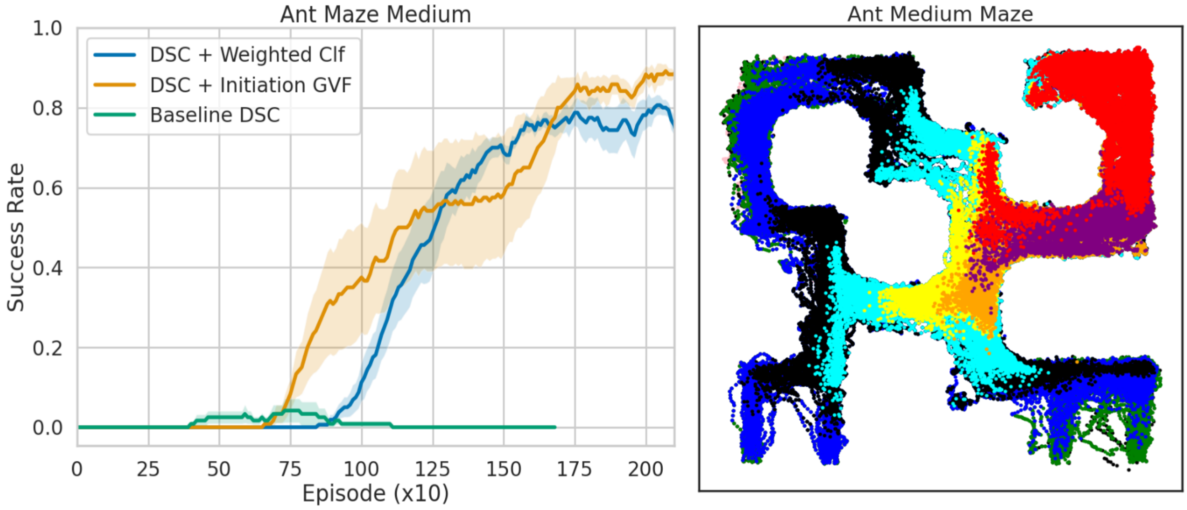


Figure 5.7: *(left)* Comparing the performance of baseline DSC (Bagaria et al., 2021a) to versions of DSC with our initiation learning techniques. Solid lines denote average success rate over 5 random seeds, shaded regions denote standard error. *(right)* A visualization of initiation sets learned in ANT MEDIUM-MAZE, where the ant robot has to navigate from the bottom-right to the top-left. Each color denotes the initiation set of a different option; although the plot only shows the location of the ant in the maze, the initiation set is learned using the full 30-dimensional state.

Experimental setup. We compare baseline DSC to versions that modify the way in which option initiation sets are learned. We use the ANT MEDIUM MAZE environment where the agent gets a sparse terminating reward of 0 for reaching the goal and -1 every other step; each episode lasts a maximum of 1000 steps (Fu et al., 2020; Todorov et al., 2012). This is a challenging problem that cannot be solved with flat RL algorithms in a reasonable amount of time. Baseline DSC was able to solve this problem by learning initiation sets defined over a subset of the state (x, y location of the ant) (Bagaria et al., 2021a), but here, we do not assume access to such privileged information—all methods learn initiation sets using a dense neural network that maps the full 30-dimensional continuous state to an initiation probability.

We used the author implementation (Bagaria et al., 2021a) where option policies are parameterized using goal-conditioned value functions (Schaul et al., 2015). When using the IVF directly as the initiation set, we mimic the structure of main agent and use a goal-conditioned value function to represent the IVF; the final initiation set is defined as

$$\mathcal{I}_o = \Big\{ s : \max_{g \in \beta_o} \mathcal{V}_\theta(s, g) > T, \forall s \in \mathcal{S} \Big\}.$$

Figure 5.7 *(left)* shows that baseline DSC is unable to solve Medium Ant-Maze; but by changing the way options learn their initiation sets, we are able to solve the problem quickly and reliably. The weighted classifier slightly underperforms the pure IVF approach in terms of mean return, but it has lower variance across runs. Figure 5.7 *(right)* visualizes the initiation sets learned by the weighted classification approach; the figure makes it clear that different learned options specialize in different parts of the maze.

5.3 Conclusion

Learning initiation sets is a critical component of skill discovery, but treating it as binary classification misses key characteristics of the problem. Specifically, it does not address the non-stationarity that results from a continually changing policy, the pessimistic bias of learning initiation sets online, or the temporal structure in the problem. To address these challenges, we proposed the Initiation Value Function (IVF), a general value function tailored specifically to learning initiation sets. We used the IVF directly via thresholding and also via a weighted binary classifier which adapts to changing option policies. Through our experiments, we showed that our proposals lead to higher quality initiation sets, can lead to faster learning of a single option and boost the performance of an existing skill discovery algorithm.

A limitation of the IVF cumulant proposed in Section 5.1.1 is that it only works for goal-reaching options. While this covers many cases, it cannot handle tasks without specific targets, such as maximizing a robot's velocity. Developing cumulants for more general option subtasks remains an open challenge.

CHAPTER 6

Skill Discovery for Exploration and Planning using Deep Skill Graphs

The skill discovery problem has been studied in many different contexts—multi-task learning (Frans et al., 2018; Veeriah et al., 2021), exploration (Machado et al., 2023; Jinnai et al., 2019), and planning(Wan and Sutton, 2022; Jinnai et al., 2018)—but how these settings culminate in a single algorithm remains unclear. We propose an algorithm that discovers skills that aid exploration, and which are subsequently useful for planning in a multi-task context. Our algorithm learns a collection of skills that form a graph, where the nodes are subgoals and the edges are the policies that navigate between them. To construct this graph, we first discover subgoal regions that optimize for coverage of the state-space, using techniques inspired by the robot motion-planning literature (LaValle, 1998) that causes the graph to grow towards large unexplored regions. These regions are subsequently connected using Deep Skill Chaining (Bagaria and Konidaris, 2020a; Bagaria et al., 2021a), resulting in a collection of skills that enable the agent to reliably move between subgoals. The chainability of the discovered skills, i.e, the property that successful execution of one permits the execution of another, allows the agent to build a graph that is suitable for deterministic planning (Konidaris et al., 2018). Finally, our

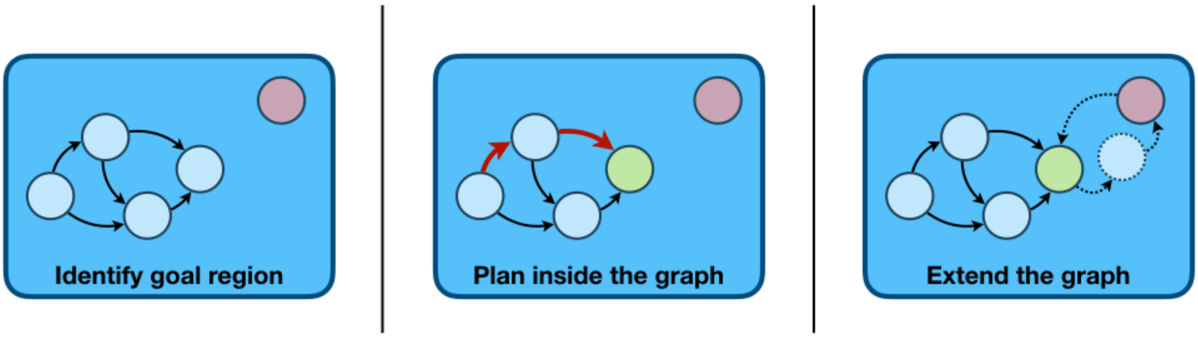


Figure 6.1: Given a goal (red) outside the graph, the agent uses the graph to plan to reach the node closest to that goal (green). It then expands the graph by constructing a skill chain connecting the goal to the graph.

algorithm operates in the multi-task setting (Taylor and Stone, 2009) because it discovers a multi-query graph (Kavraki et al., 1996) that it can use to plan between arbitrary start-goal pairs at test time. At test time, if a goal state lies inside the skill-graph, the agent can reach it without any further learning; if not, the agent plans to the nearest subgoal and then switches to learning a new skill chain that extends the graph to reach its new goal.

We test our algorithm on four maze-navigation tasks in MuJoCo (Todorov et al., 2012), where it outperforms flat model-free RL (Andrychowicz et al., 2017), model-based RL (Nagabandi et al., 2018) and state-of-the-art skill-discovery algorithms (Levy et al., 2019; Sharma et al., 2020c).

6.1 Deep Skill Graphs

To solve goal-reaching tasks, options must be sequentially composable: the agent must be able to successfully execute one option after another until it reaches the goal. Sequential composition is possible if and only if successful execution of one option leads the agent to a state that permits the execution of another.

When the agent need only ever navigate from a single given start state to a single given goal state, it must therefore learn options that form a *chain* connecting those two states.

In the more general case where the agent is given a single goal that must be reachable from *any* start state, it must instead build a *tree* of options, enabling it to follow a chain from any state to the goal. In the most general case, the agent may be asked to reach *any* goal state from *any* start state; since that requires a chain of options connecting *any* possible start-goal pair, the agent's options must necessarily be organized into a *graph*. We therefore propose to explicitly construct options so that they form such a graph.

6.1.1 Definitions

Before describing how the skill graph is constructed and used, we first define its key components:

Discovered goal regions. The DSG agent proposes its own goals during an unsupervised training phase. For each proposed goal state g, we define a goal region $\epsilon_g : \mathcal{S} \rightarrow \{0, 1\}$ as an ϵ-ball centered around g. $\mathcal{B} = \{\epsilon_{g_1}, .., \epsilon_{g_n}\}$ is the set of all such goal regions. To unify notation with options, we additionally define $\mathcal{I}_{\epsilon_g} = \epsilon_g$, and $\mathcal{E}_{\epsilon_g}$ as the set of states in which goal region ϵ_g was actually reached.

Skill graph. A skill graph is a weighted, directed graph $\mathcal{G} = (\mathcal{V}, E, \mathcal{W})$. Each vertex $i \in \mathcal{V}$ either corresponds to an option or a goal region. Edge $e_{i \rightarrow j}$ exists between vertices i and j if and only if the effect set of i is inside the initiation set of j, i.e, $\mathcal{E}_i \subseteq \mathcal{I}_j$ (Konidaris et al., 2018). The edge weight $w_{i,j} \in \mathcal{W}$ is the reward accumulated by going from vertex i to j.

Mapping states to vertices. Given agent state s, $\mathcal{V}(s)$ denotes the set of vertices in the skill graph that s maps to. First, we find the set of options in the graph that the agent can execute at s:

$$\mathcal{O}(s) = \{o \mid \mathcal{I}_o(s) = 1, \forall o \in \mathcal{O}\}. \tag{6.1}$$

Then, we enumerate the set of goal regions that s satisfies:

$$\mathcal{B}(s) = \{\epsilon_g \mid \epsilon_g(s) = 1, \forall \epsilon_g \in \mathcal{B}\}. \tag{6.2}$$

Finally, the set of vertices that s maps to is given by:

$$\mathcal{V}(s) = \mathcal{O}(s) \cup \mathcal{B}(s). \tag{6.3}$$

The **descendants** $\mathcal{D}(v)$ of vertex v is the set of vertices reachable from v. The **ancestors** $\mathcal{A}(v)$ of v are the vertices from which v is reachable via $\mathcal{G}$:

$$\mathcal{D}(v) = \{v' \mid \mathcal{G}.\texttt{has-path}(v, v'),\ \forall v' \in \mathcal{V}\}, \tag{6.4}$$

$$\mathcal{A}(v) = \{v' \mid \mathcal{G}.\texttt{has-path}(v', v),\ \forall v' \in \mathcal{V}\}. \tag{6.5}$$

6.1.2 Overview of the Graph Construction Algorithm

During training, the agent alternates between (a) expanding the graph into unexplored regions and (b) increasing the connectivity of the existing graph. The high-level algorithm for both cases is similar; its three steps are illustrated in Figure 6.1.

The details of *how* each step is performed differs based on whether the agent is trying to expand the graph or consolidate it. During graph expansion, the agent generates new goals outside the graph (Section 6.1.3), navigates to its nearest neighbor in the graph (Section 6.1.3) and then moves K steps in the direction of the goal (Section 6.1.3).

During graph consolidation, the agent instead picks a node from the closest unconnected sub-graph as the goal (Section 6.1.4), navigates to its nearest neighbor in the current sub-graph (Section 6.1.4) and then learns an option chain that connects the sub-graphs (Section 6.1.4).

6.1.3 Graph Expansion

Every few episodes, the agent expands the skill-graph to new regions of the state-space.

Generate New Goals

Regions covered by the graph represent places where the agent has already achieved mastery (Kaelbling, 1993; Veeriah et al., 2018) because the agent can plan using learned skills to reliably reach arbitrary states inside the graph. To increase the portion of the world over which it has achieved mastery, the skill-graph should expand into largely unexplored regions of the state-space.

In other words, given the current graph $\mathcal{G}$, the agent should generate a goal state g such that $\mathcal{G}$ grows towards the largest region it has not yet covered. As Lindemann and LaValle (2004) (somewhat surprisingly) showed in the context of motion planning, randomly sampling g from the state-space does exactly that. So, we set $g = s_{rand} \sim \mathcal{S}$.[1]

Identify the Nearest Neighbor

Given a randomly sampled goal state s_{rand} that lies outside the graph, the agent identifies its nearest neighbor in the graph v_{nn}. To find v_{nn}, it first uses Equation 6.4 to instantiate the set of vertices reachable from the current state: $\mathcal{D}(s_t) = \{\mathcal{D}(v) | v \in \mathcal{V}(s_t)\}$. Then, it finds the vertex in $\mathcal{D}(s_t)$ closest to s_{rand} and sets v_{nn} to it:

$$\begin{aligned} v_{nn} &= \underset{v \in \mathcal{D}(s_t)}{\arg\min} \, ||s_{rand} - v||^2 \\ &= \underset{v \in \mathcal{D}(s_t)}{\arg\min} \left[\max_{s \in \mathcal{E}_v} ||s_{rand} - s||^2 \right]. \end{aligned}$$

Note that, in the equation above: (a) we define the distance between vertex v and state s_{rand} as the maximum distance between any of the states in the effect set of v and s_{rand}

---

[1] This assumes that we can sample states from the environment. In MuJoCo, where the observation space is factored and bounded, this is straightforward. Sampling from pixel-based observation spaces is more involved, and is the subject of active research (Nair et al., 2018; Pong et al., 2019).

and (b) we use the Euclidean metric and leave discovering more sophisticated metrics (Mahadevan and Maggioni, 2007; Hartikainen et al., 2020; Pitis et al., 2020a) for future work.

Navigate to the Nearest Neighbor

Given that the agent is at state s_t and wants to reach vertex v_{nn}, it must decide which option to execute. There are two possible cases: (a) if there is a path from $\mathcal{V}(s_t)$ to v_{nn} in $\mathcal{G}$, the agent uses Equation 6.1 to find $\mathcal{O}(s_t)$, the set of options available in s_t. Then, it uses graph search (Dijkstra, 1959) to find a plan from each $o \in \mathcal{O}(s_t)$ to v_{nn}, selects the least-cost plan, and executes its first option. Having landed in a new state $s_{t+\tau}$, it re-plans to go from $s_{t+\tau}$ to v_{nn}. This process continues until it reaches v_{nn}. (b) If there is no such path on the graph, the agent uses DSC's model-free policy over options $\pi_{\mathcal{O}}$ to select options (or primitive actions). Since DSC trains this policy using SMDP Q-learning, it is not as effective as using the planner for option selection.

Extend the Graph

Having reached v_{nn}, the agent attempts to extend the graph towards s_{rand}. However, s_{rand} may be unreachable from v_{nn} either because it is inside an obstacle or because of the limits of the agent's current policy. As a result, s_{rand} itself cannot be used as a goal state. In RRT, this issue is resolved by using the given dynamics model to move K steps in the *direction* of s_{rand}. Since a dynamics model is not typically known a priori in the RL setting, we learn one from the data collected during training and use the learned model in conjunction with Model Predictive Control (MPC) (Garcia et al., 1989) to move in the direction of s_{rand}. The state that the model-based controller ends up in, s_{MPC}, becomes a new goal state and $\epsilon_{s_{MPC}}$ is added to the list of goal regions $\mathcal{B}$. An example of this process is shown in Figure 6.3c.

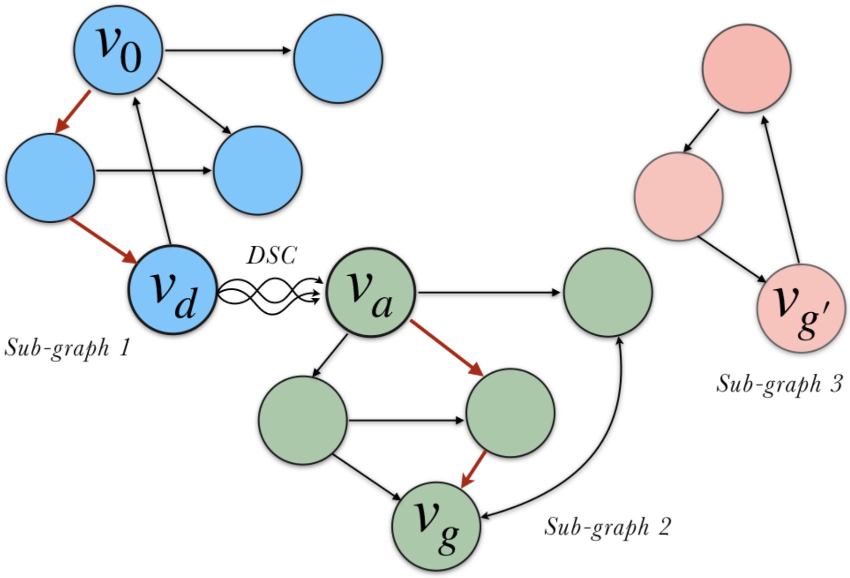


Figure 6.2: When an agent aims to consolidate the graph at training time and is in graph vertex v_0, it targets sub-graph 2 because it is closer than sub-graph 3. Then it identifies the closest descendant-ancestor pair (v_d, v_a). It uses its planner to traverse $v_0 \rightarrow v_d$ and $v_a \rightarrow v_g$. Since it does not have skills between v_d and v_a, it uses DSC to learn a chain of skills connecting sub-graphs 1 and 2.

6.1.4 Graph Consolidation

For training episodes in which the agent is not expanding the graph, it improves the graph's connectivity by selecting an existing node and attempting to reach it.

Goal Identification

To inform the decision about which node to target, the agent identifies parts of the graph it currently has difficulty reaching. To do this, it first enumerates the vertices to which there is *no path* from its current state s_t. To pick a specific one to target, it uses a simple greedy heuristic: target the nearest unconnected sub-graph to greedily maximize the connectivity of the overall graph (Kuffner and LaValle, 2000). This process is illustrated in Figure 6.2.

Identify the Nearest Neighbor

Having selected a goal vertex v_g, DSG identifies its current sub-graph (the set of descendants $\mathcal{D}(s_t)$), and the sub-graph from which it can reach the goal vertex v_g (the set of ancestors $\mathcal{A}(v_g)$). Finally, it identifies the pair of nodes from $\mathcal{D}(s_t)$ and $\mathcal{A}(v_g)$ closest to each other:

$$\begin{aligned} v_d, v_a &= \underset{\substack{v_d \in \mathcal{D}(s_t), \\ v_a \in \mathcal{A}(v_g)}}{\arg\min} ||v_d - v_a||^2 \\ &= \underset{v_d, v_a}{\arg\min} \left[\max_{\substack{s_d \in \mathcal{E}_{v_d}, \\ s_a \in \mathcal{E}_{v_a}}} ||s_d - s_a||^2 \right]. \end{aligned}$$

Extend the Graph

The agent uses its planner to reach v_d (using the procedure described in Section 6.1.3), DSC to reach v_a from v_d, and the planner again to finally reach v_g. This procedure minimizes the number of new skills required to connect the two sub-graphs. Once these skills have been learned, there is a path in $\mathcal{G}$ from s_0 to v_g, and the agent can henceforth plan to target v_g.

Adding Edges to the Graph. For deterministic plans in the skill-graph to correspond to feasible solutions in the ground MDP, Konidaris et al. (2018) showed that any two options o_1, o_2 can have an edge $e_{1,2}$ between them if and only if there is a guarantee that successful execution of o_1 will allow the agent to execute o_2, i.e, $e_{1,2}$ exists iff $\mathcal{E}_{o_1} \subseteq \mathcal{I}_{o_2}$. To implement this rule, we store all the states in which o_1 successfully terminated and check if all of them lie inside $\mathcal{I}_{o_2}$. Similar logic can be applied to creating edges between options and goal regions in the graph.

6.1.5 Querying the Skill Graph at Test Time

Given a task expressed as a start state s_0 and a goal region ϵ_g at test time, the agent can use the graph as follows:

- if ϵ_g completely contains the effect set of an option, reaching that effect set implies reaching ϵ_g, so the agent plans to reach g without any additional learning.
- if the goal outside the graph, DSG extends the graph just as it did during training: it plans to the nearest node and then uses DSC to build a chain of options from the graph to reach g.
- If the start state s_0 itself lies outside the graph, DSG additionally creates a skill chain from s_0 to its nearest node in the graph.

6.2 Experiments

We tested DSG on the continuous control tasks shown in Figure 6.5. These tasks are adapted from the "Datasets for RL" benchmark (Fu et al., 2020)[2] and are challenging for non-hierarchical methods, which make little-to-no learning progress (Duan et al., 2016). Since we already assume a Euclidean distance metric in Section 6.1.3, we use the dense reward version of these tasks, i.e, $\mathcal{R}(s, g) = -||s - g||$. Video and code can be found on our website.

6.2.1 Implementation Details

We follow Nagabandi et al. (2018) to learn a model-based controller to use as option policies $\pi_o, \forall o \in \mathcal{O}$. Specifically, transitions seen by all options are used to train a neural dynamics model f_ξ, which is then used to select actions by approximately solving the

---

[2] We use the mazes from this suite, not the demonstration data.

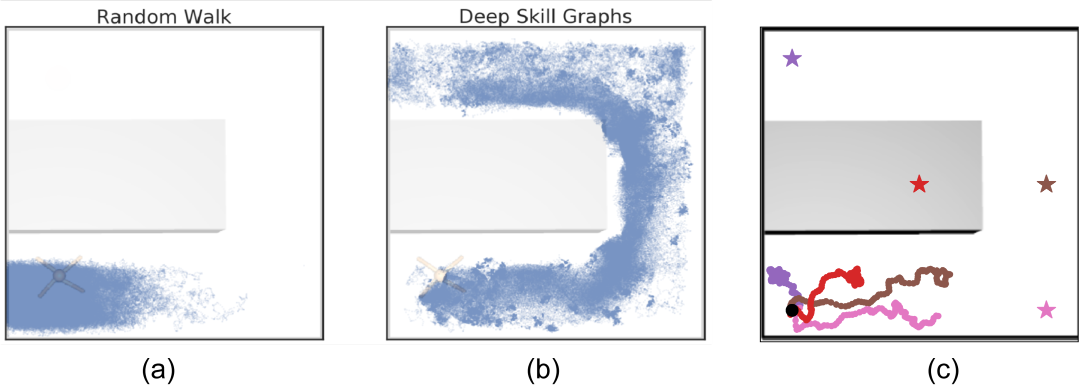


Figure 6.3: (a) States visited under a random walk and (b) using DSG. (c) model-based extrapolation for discovering goals during unsupervised training: the stars represent randomly sampled states; trajectories of the same color target the respective goals using MPC. All data is collected in Ant U-Maze, where the agent starts at the bottom-left of the maze.

following H-horizon optimization problem:

$$\pi_o(s|g) = \underset{a \in \mathcal{U}^{|\mathcal{A}|}(-1,1)}{\arg\max} \sum_{t=1}^{H} \gamma^{t-1} \mathcal{R}(s_t, g), \tag{6.6}$$

such that $s_{t+1} = f_\xi(s_t, a)$. The solution to Equation 6.6 is a sequence of actions; rather than executing this sequence open-loop, we use MPC to execute the first action and re-plan at the next time-step (Nagabandi et al., 2018). Before executing option policy π_o, we randomly sample from the option's subgoal region, i.e, $g \sim \beta_o$ via the option's effect set.

6.2.2 Qualitative Evaluation

Exploration Property of DSG. We compare the states visited by the DSG agent and those visited under a random walk. Figure 6.3 shows that in the Ant U-Maze environment, skills can lead to temporally extended exploration as opposed to the dithering behavior of random walks, even in the absence of an extrinsic reward function.

Incremental Graph Expansion. Figure 6.4 provides some intuition on why DSG can effectively explore large regions of the state-space—the skill-graph begins at the start

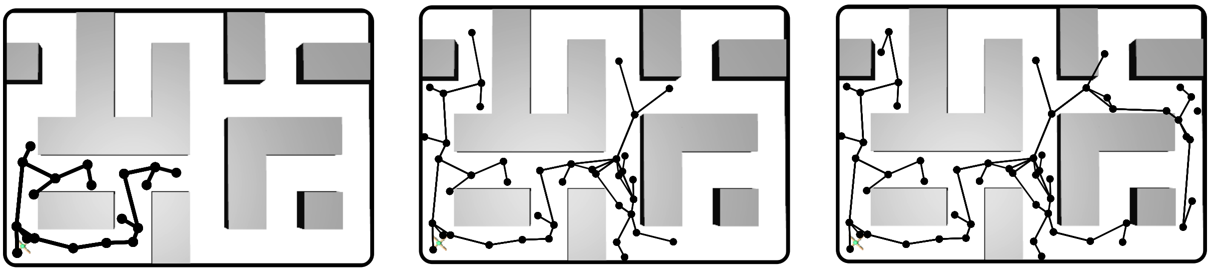


Figure 6.4: Incremental expansion of the skill graph towards unexplored regions in the large ant-maze. Although the skill-graph is directed, we visualize it as un-directed for simplicity. To visualize option nodes, we plot the median state of its effect set.

state and incrementally expands into unexplored regions of the state-space. By planning and executing learned skills inside the graph, the agent can reliably get back to the frontier of its knowledge (as represented by the outermost vertices in the graph). Exploration from the frontier in turn allows it to amass the experiences it needs to further expand the graph. By interleaving planning and chaining in this way, the DSG agent incrementally achieves coverage of ever increasing portions of the state-space

6.2.3 Quantitative Evaluation

Experimental Setup. The agent discovers skills during an unsupervised training phase (which lasts for 1000 episodes in Reacher and U-Maze, 1500 episodes in Medium-Maze and 2000 episodes in the Large-Maze). During this period, its start-state is fixed to be at the bottom-left of every maze. At test time, we generate 20 random start-goal pairs from the maze and record the average success rate (Andrychowicz et al., 2017) of the agent over 50 trials per start-goal pair (Sharma et al., 2020c). All competing methods are tested on the same set of states.

Comparisons. We compare DSG to the following

1. Flat model-free baselines: **HER** (Andrychowicz et al., 2017) learns policies that generalize across goals by taking the goal state as an additional input during training. Because of their widespread use, we use them as a representative non-hierarchical

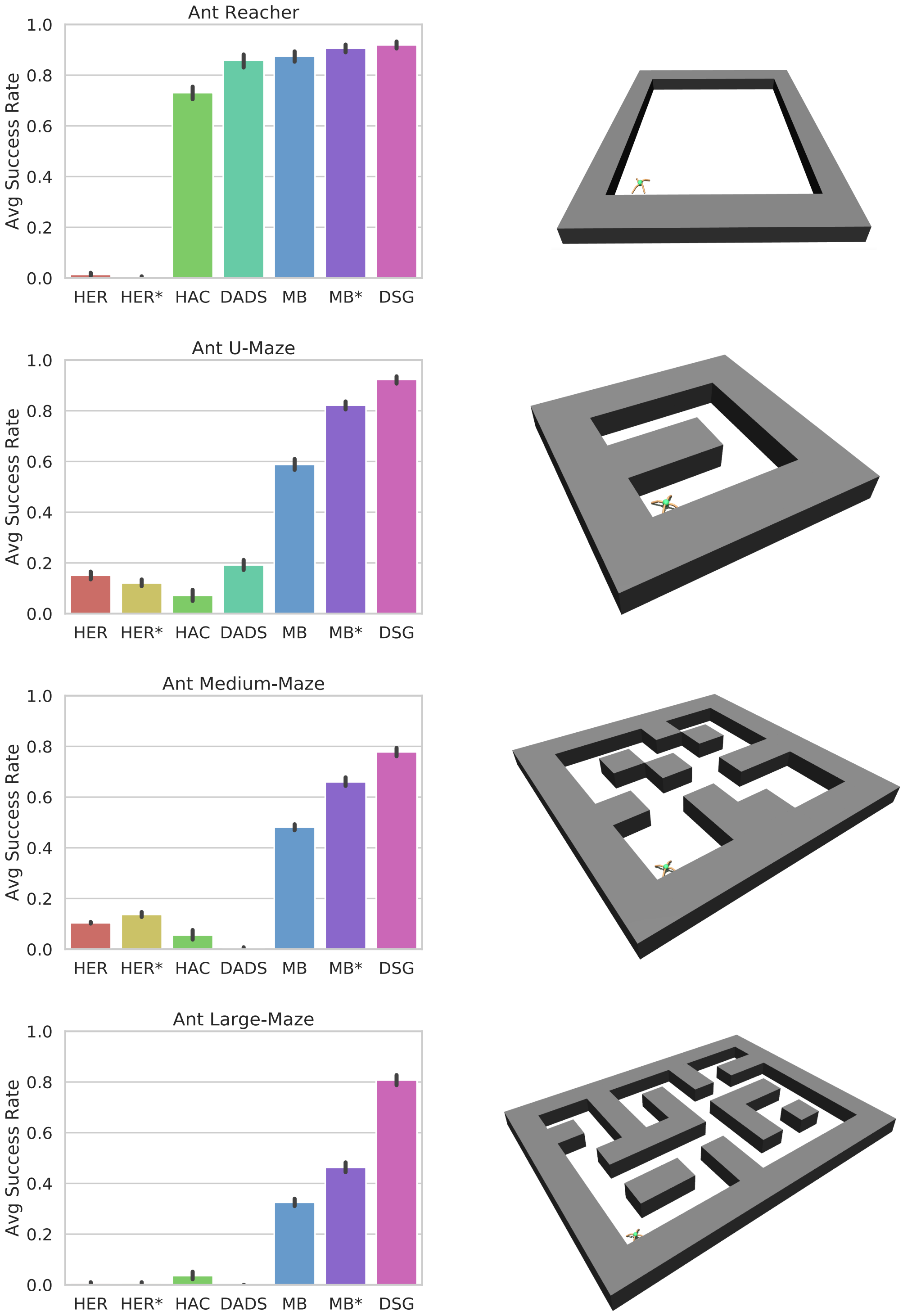


Figure 6.5: Test-time success rates averaged over 20 randomly sampled start-goal pairs and 50 trials; error bars denote standard deviation. All algorithms were trained for the same number of episodes prior to being tested—their training performance is not shown here.

baseline. Since HER does not have an explicit mechanism for exploration, we additionally favorably expand its start-state distribution during training to include all valid locations in the maze—we call this baseline **HER***. HER policies are learned using TD3 (Fujimoto et al., 2018).

2. Flat model-based baselines: Since we use the model-based **(MB)** controller from Nagabandi et al. (2018) as option policies, we include it as our model-based baseline. Similar to HER*, we include a version **MB*** which also gets favorable access to a wide start-state distribution.

3. Feudal HRL baseline: **HAC** (Levy et al., 2019) is a hierarchical extension of HER, and has achieved state-of-the-art performance on similar tasks.

   HER, HER*, MB, MB* and HAC sample goals uniformly from the free-space in the maze during training.

4. Empowerment HRL baseline: To compare against **DADS**, we trained the skill-primitives in the Ant-Reacher environment and used them at test-time in the different mazes. This strategy outperformed that of discovering skills in each specific maze environment, since there was no space for the DADS agent to learn diverse skills.

6.3 Results

Figure 6.5 shows that DSG comfortably outperforms all baselines. Our results confirm that while in principle flat goal-conditioned RL can solve the kind of goal-reaching tasks we consider, they require orders of magnitude more data and struggle to generalize to distant goals.

Both HAC and DADS were able to achieve comparable (and good) results in Ant-Reacher (which was the task that they were evaluated on in their respective papers). However, unlike DSG, they struggled to maintain that level of performance as the problem

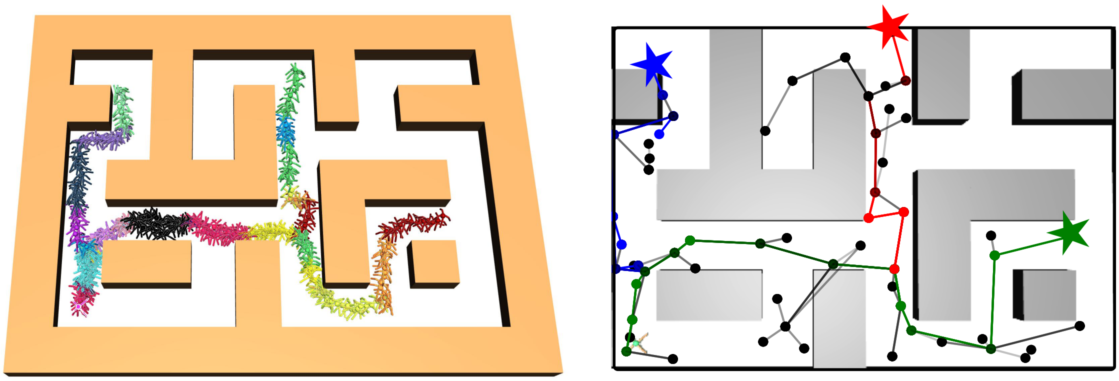


Figure 6.6: Discovered graph-based abstraction *(right)* of a high-dimensional continuous control problem *(left)* during an unsupervised training phase. When given test-time goals (stars), the agent finds the high-level solution trajectories on the graph, which are shown in red, green and blue *(right)*. Then the high-level solutions are translated to joint torques using discovered skills, which are shown as different colors *(left)*.

got harder with larger, more complicated mazes. HAC implicitly assumes that the space of subgoals is smooth—which is violated in our mazes because several subgoal proposals made by the manager correspond to states that are inside obstacles, and hence unreachable by the workers. DADS discovers maximally diverse skills, but solving mazes requires sequencing a very specific sequence of actions (e.g., "go right until you can go left")—a property not captured by their diversity objective. Compared to DSG, an advantage of DADS is that their skills are *portable* (Konidaris and Barto, 2007), meaning that the same skill can be transferred to different regions of the state-space.

Upon examining the goal states that DSG *failed* to reach at test-time, we found that these failures could be attributed to our reliance on dense rewards. Specifically, if the goal state lay on the other side of an obstacle from its nearest node in the graph, DSG would be unable to extend the graph towards this goal. It is well known that dense reward functions can lead to sub-optimal policies (Randløv and Alstrøm, 1998; Mataric, 1994), and extending our algorithm to the sparse-reward setting should mitigate these failures.

6.4 Conclusion

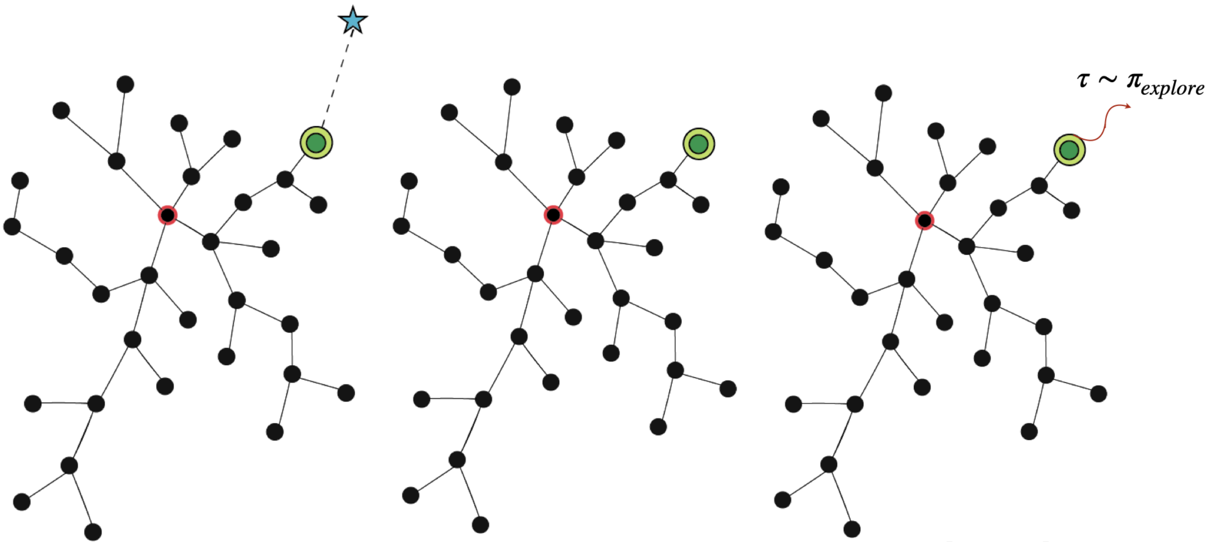
We introduced an algorithm that builds a graph abstraction of large continuous MDPs. Like all RL agents, DSG starts by exploring the environment since it does not know enough about it to plan. Then, it proposes its own goals and learns skills to reliably reach those goals. Together, proposed goals and skills enable high-level planning.

We showed that skill graphs grow outward from the start-state towards large unexplored regions, reflecting mastery over ever increasing portions of the state-space. Finally, we tested our algorithm on a series of maze navigation tasks and showed that DSG can be used to reach novel goals at test-time in a small number of trials. We compared the zero-shot generalization capability of our algorithm to that of popular flat and state-of-the-art hierarchical alternatives and showed that DSG significantly outperforms them.

CHAPTER 7

Intrinsically Motivated Discovery of Temporally Abstract World Model Graphs

A recurring theme of this thesis has been the desire for learning plannable models of the world that are abstract in both state and time. In Chapter 6, we discussed the Deep Skill Graphs (DSG) algorithm and how it builds a graph-based abstraction of large continuous environments—nodes in the graph represent abstract states and edges represent skill policies. The usefulness of the skill graph stems primarily from its *expansion property*: over time, the graph expands outward from the start-state towards unexplored regions (LaValle, 1998), allowing the agent to gradually increase mastery over its environment (Veeriah et al., 2018). Graph expansion in DSG is achieved via random sampling from the state-space and nearest-neighbor search: operations that are intractable in environments with high-dimensional observations (e.g, images). Furthermore, the DSG algorithm attempts to increase the probability of *all* edges (by repeatedly executing the corresponding skills) so that the resulting graph is robust and reliable *everywhere.* However, when the environment is much larger than the agent, exhaustive coverage is intractable (Sutton et al., 2022),



$n_{new} \sim \mathcal{S}$

(a)

$n_{expansion} = \mathbf{argmax}_n \mathbb{E}_s\left[V^{\theta}_{explore}(s)\right]$

(b)

$n_{new} \leftarrow \max_{s \in \tau}\left[R^{\phi}_{explore}(s)\right]$

(c)

Figure 7.1: (a) In the original DSG algorithm, a state is sampled uniformly at random (blue star) from the state-space $\mathcal{S}$ and the graph is pulled towards it via its nearest neighbor node (green). (b) In the new algorithm, IM-DSG, we identify a node to expand using an exploration value function V_{explore}. (c) Once the agent reaches the expansion node, it executes the exploration policy π_{explore}, and the most salient state in the resulting trajectory τ is identified as a target for a new skill.

and the agent should prioritize practising skills that are more likely to result in higher reward (Sutton et al., 2024).

How can we incrementally build the skill graph so that it retains its expansion property, but does not require a state-sampler or nearest-neighbor search to do so? Instead of generating a random state and *pulling* the graph towards it (see Figure 7.1(a)), we use ideas from intrinsic motivation (Barto, 2013; Schmidhuber, 2010b) to identify an existing node for expansion and *push* the graph outward from it (Figure 7.1(b); Lindemann and LaValle 2004). The agent plans with its existing skills to reach the expansion node, and then executes a *low-level exploration policy* trained using RL to maximize a novelty-based intrinsic reward (Sutton, 1990; Taïga et al., 2019). Since the intrinsic reward is likely to be higher in states that are outside the graph, executing the exploration policy will result in a trajectory that will tend to move away from the existing graph, i.e., towards those states where the agent does not yet have a reliable abstract model (Figure 7.1(c)). The agent then identifies the most interesting state (Chentanez et al., 2005) in that trajectory and builds a new skill to target it (Şimşek and Barto, 2004). Over time, the agent refines the skill that reaches this new node in the graph, and in doing so, converts this once novel region into an area over which it has gained mastery via its abstract model (Veeriah et al., 2018). We call this algorithm Intrinsically Motivated Deep Skill Graphs (IM-DSG).

We test IM-DSG in a series of exploration problems with image-based observation spaces, to which DSG cannot be applied. Our qualitative results show that IM-DSG discovers options that achieve intuitively meaningful subgoals in the environment and that the acquired skill graph incrementally expands outward from the start state. Quantitatively, we show that our method outperforms pure novelty-maximization techniques (Taïga et al., 2019) as well as baseline model-free RL (Kapturowski et al., 2019) in MiniGrid (Chevalier-Boisvert et al., 2023), Visual Taxi (Dietterich, 2000; Diuk et al., 2008; Allen et al., 2021), and Sokoban (Schrader, 2018); finally we show that the learned abstract model is useful in a multi-task context to achieve held-out goals in a zero-shot fashion in Visual Pinball (Konidaris and Barto, 2009b; **?**).

7.1 Background and Related Work

We model the interaction of the agent with its environment as a Markov Decision Process $M = (S, A, R, T, \gamma)$ (Sutton and Barto, 2018). To aid in reward maximization, our agent will discover options $o \in \mathcal{O} = (\mathcal{I}_o, \pi_o, \beta_o)$ (Sutton et al., 1999), where $\mathcal{I}_o(s)$ is the probability of initiating option o in state s, π_o is a policy which outputs primitive actions and $\beta_o : s \to \{0, 1\}$ denotes the termination region of the option, which we overload with the subgoal of the option. Additionally, each option has a maximum horizon of H steps and has en effect set Eff(o), which denotes the set of states in which the option triggered its subgoal condition β_o (Konidaris et al., 2018).

Universal Value Function Approximators (UVFAs) (Schaul et al., 2015) provide a scalable way to learn goal-conditioned value functions $V_g : S \to \mathbb{R}$, $Q_g : S \times A \to \mathbb{R}$ and their corresponding goal-conditioned policies $\pi_g : S \to A$ (Kaelbling, 1993) using function approximation. We use UVFAs to parameterize option value functions and policies: instead of representing each option's policy separately (Sutton et al., 2011), all options can condition the same function approximator on its own subgoals (Bagaria et al., 2021a): $\pi_o(s) = \arg\max_a Q_g(s,a) = \arg\max_a Q_{\beta_o}(s,a)$.

Exploration in RL. Exploration is a fundamental problem in RL. In the multi-armed bandit setting, the agent can pick actions that maximize the upper-confidence bound (UCB) of the action-value function: $\pi(a) = \arg\max_{a \in A} \left[Q(a) + \frac{c}{\sqrt{N[a]}} \right]$ to optimally trade-off exploration and exploitation (Lattimore and Szepesvári, 2020). In tabular RL, the agent's uncertainty about its transition and reward function (together called a "model") can once again be captured using the proxy of visitation counts over the state-action space: as long as the bonus reward (Sutton, 1990) for trying action a from state s decays as $\frac{1}{\sqrt{N[s,a]}}$, the agent can rapidly learn the optimal policy π^* (Strehl and Littman, 2008). When the environment is too big to permit tabular counting, the notion of counts can be generalized to "pseudocounts" $\mathcal{N}(s)$ (Bellemare et al., 2016): states that are similar to one another have similar pseudocounts. Coin Flip Networks (CFN) (Lobel et al., 2023) is a simple, principled and state-of-the-art technique which uses neural networks ϕ to predict pseudocounts in high-dimensional state-action spaces. Predicted pseudocounts are turned into an intrinsic reward function $r^{int}(s) = \frac{1}{\sqrt{N_\phi(s)}}$ and a policy is learned using off-the shelf model-free RL algorithms to maximize the sum of intrinsic and extrinsic rewards (Taïga et al., 2019). Novelty seeking agents exhibit impressive performance in Atari games (Kapturowski et al., 2022; Ecoffet et al., 2019), but they do not learn a model suitable for high-level planning. In this chapter, we propose an algorithm that combines novelty maximization objectives (which we refer to as "low-level exploration") with a drive to learn an abstract, plannable model of the world ("high-level exploration").

Model-based RL. The IM-DSG algorithm introduced in this chapter is closely related to several classic model-based RL algorithms. Chief among them is the E^3 algorithm, which divides the state-space into 3 sets: known, unknown but visited, and unvisited. Conceptually, the known states are those in which the agent has a strong model over; this is exactly what the skill-graph also represents. The states that are "unknown but visited" are analogous to the leaves of the skill graph (from the perspective of the start-state), which serve as expansion nodes. Finally, the expansion of the skill graph is designed so that more states from the unvisited region are eventually moved into the known region. As we will see in Section 7.2, we take inspiration from algorithms that exploit the idea of *optimism under uncertainty* such as R-Max (Brafman and Tennenholtz, 2002), UCRL2 (Auer et al., 2008) and MBIE-EB (Strehl and Littman, 2008) to carefully balance exploration and exploitation. Unlike those algorithms, we do not simply take random actions in the unvisited region, but instead follow a policy that demonstrates *deep exploration* (Osband et al., 2016a) because it might not be trivial to reach interesting states (for example, refer to the "narrow passage" problem in motion planning (Kavraki et al., 1996)).

Deep Model-based RL. Dreamer (Hafner et al., 2018) and MuZero (Schrittwieser et al., 2021) family of algorithms learn a physics-based model of that environment for planning. These algorithms have demonstrated impressive results in a variety of domains (Hafner et al., 2023). However, all learning (even in LEXA (Mendonca et al., 2021) and Director (Hafner et al., 2022), which learn temporal abstractions) is predicated on accurately approximating **one-step** transition and reward models in the agent's native state-action space. One-step models suffer from the curse of the horizon (Sutton et al., 1999): model prediction errors compound over-time and complicate exploration and credit assignment (Talvitie, 2017). Our work seeks to bypass the challenges of model-learning in the raw state-action space: model-free policies are learned in the low-level and temporally-extended models are learned and used in the high-level.

Go-Explore shares a similar motivation to our algorithm: exploration is more effective when initiated from the frontiers of the agent's experience. The initial version of Go-Explore relied on simulator resets, which is a limiting assumption (Ecoffet et al., 2019). More recent versions do not rely on resets, but require a pre-specified "cell" representation and exploit the near-determinism of Atari domains (Machado et al., 2018a) by using imitation learning to return to the frontier. Crucially, Go-Explore does not address the challenges of learning a world model and planning with it while doing exploration, which is the focus of our work.

Graph-based HRL methods. Since DSG (Bagaria et al., 2021b), some papers (Lee et al., 2022; Lo et al., 2022; Roice et al., 2024) have also built graph-based abstractions of the environment, but the methods do not generalize to image-based observation spaces. Some papers provide ways to learn abstract models (Konidaris et al., 2018; Lo et al., 2022; Mann et al., 2015; Sutton et al., 2024), but they assume that options are given to the agent. Other methods (Eysenbach et al., 2019b; Huang et al., 2019; Campos Camúñez et al., 2020; Nasiriany et al., 2019; Sharma et al., 2020c) decompose the agent's training into a pre-training stage and a planning phase: a goal-conditioned value function is learned over the entire domain and *then* planning is done to improve the effectiveness of the pre-trained value functions; this approach is insufficient in the continual learning setup (Ring, 1994) where the agent may never visit all parts of the environment, especially without using a planner in the loop. Another common assumption made by existing graph-based HRL methods is that of uniform resets: the agent is randomly reset to a new location at the beginning of every episode (Eysenbach et al., 2019b; Huang et al., 2019; Zhang et al., 2021b); this bypasses the exploration problem, which is a key motivation for our work.

7.2 Intrinsically Motivated Deep Skill Graphs

Exploration and planning are deeply intertwined. Exploration aids planning by preferentially seeking out data from regions where the model is currently too weak to support effective planning. But, planning also supports effective exploration: the agent can plan with its abstract model so that it can reach interesting areas reliably, and then conduct local exploration in a more targeted fashion. The Intrinsically Motivated Deep Skill Graphs (IM-DSG) agent interleaves exploration and planning in this way. To support planning, it builds a graph-based model of the environment. Exploration proceeds both inside the graph (to strengthen the existing model) and outside it (to increase the region over which the agent can plan). Inside the graph, the agent refines skills that are currently weak and could lead to high reward; outside the graph, the agent uses a novelty and reward maximizing policy to search for new salient regions that can be reached reliably in the future using its model.

7.2.1 Semantics of the Skill Graph

Before discussing our algorithm for discovering options and planning with them, we first outline the major components of the skill graph:

Nodes and Edges. The skill graph is a weighted and directed graph $\mathcal{G} = (\mathcal{N}, E, W)$. Each **node** $n \in \mathcal{N}$ represents the terminating states of an option o_n—i.e., its subgoal—determined by the termination condition $\beta_{o_n} : s \rightarrow \{0, 1\}$. For each node n, we can measure the extent to which a state s belongs to n by considering the effect of executing option o_n from any state in the state space:

$$n(s) := \text{Eff}(o_n)(s) = \mathbb{P}(s \mid o_n); \tag{7.1}$$

that is, the distribution of states after executing o_n. Moving forward, we slightly abuse notation by taking expectations with respect to the states of a node, $\mathbb{E}_{s\sim n}[\cdot]$ to mean expectations taken with respect to the effect distribution $\mathbb{E}_{s\sim\text{Eff}(o_n)}[\cdot]$.

An **edge** $e_{n_i \to n_j} \in E$ exists between nodes n_i and n_j if executing the option policy $\pi_{o_{n_j}}$ from states in node n_i is likely to reach node n_j (Konidaris et al., 2018). The weight of an edge is the probability of successfully traversing it in a single option execution, which is equivalent to the initiation probability of option o_j from states inside node n_i: $w_{n_i \to n_j} = \mathbb{E}_{s\sim n_i}\Big[\mathcal{I}_{o_{n_j}}(s)\Big]$ (Bagaria et al., 2023).

Mapping from states to nodes and back. Given a state s, $\mathcal{N}(s)$ denotes the set of nodes that s maps to:

$$\mathcal{N}(s) = \{n \in \mathcal{N} \mid \beta_{o_n}(s) = 1\}. \tag{7.2}$$

Of course, it is possible that a state does not satisfy *any* option's termination condition; so, we augment the nodes in the graph with a null vertex φ such that $\mathcal{N}(s) = \{\varphi\}$ for such states; informally, φ represents "falling off the graph".

The descendants $\mathcal{D}(n)$ of a node n is the set of nodes that can be reached from n with nonzero probability, while staying inside the graph:

$$\mathcal{D}(n) = \{n' \in \mathcal{N} \mid \mathcal{G}.\texttt{has-path}(n, n')\}, \tag{7.3}$$

where $\mathcal{G}.\texttt{has-path}(n, n')$ is a sub-routine implemented using breadth-first search on the adjacency matrix representation of the graph $\mathcal{G}$. By combining Equations 7.2 and 7.3, we can enumerate the nodes reachable from the current state s_t as $\mathcal{D}(s_t) = \cup_{n \in \mathcal{N}(s_t)} \mathcal{D}(n)$.

7.2.2 Graph Construction Algorithm

At a high-level, the graph construction algorithm proceeds as follows: the IM-DSG agent first identifies which node in the graph to expand (Section 7.2.2.1), then it constructs (Section 7.2.2.4) and solves an abstract MDP (Section 7.2.2.5), which yields a policy over options that guides the agent to the expansion node. After following this abstract policy and reaching the expansion node, the agent executes a low-level exploration policy (Section 7.2.2.6), and finally extracts a new node from the resulting trajectory to add it to the skill graph (Section 7.2.2.7).

7.2.2.1 Identifying the Expansion Node

To select an expansion node, we need to quantify how much each node would contribute to graph expansion: nodes that cause the graph to extend further should have a higher probability of being selected for expansion. To capture this idea, we first construct an intrinsic reward function that is higher the further away a state is from the graph. But rather than relying on distance functions, which are usually unavailable in high-dimensional observation spaces, we use pseudocounts (Bellemare et al., 2016): the further away a state is from the graph, the fewer times it has likely been visited. We use Coin Flip Networks (CFN) (Lobel et al., 2023) to estimate pseudocounts and use it as our intrinsic reward:

$$r_{\text{novelty}}(s) = \frac{1}{\sqrt{N_\phi(s)}},$$

where $N_\phi(s)$ is the pseudocount of state s as approximated using CFN parameters ϕ.

We now have an intrinsic reward function r_{novelty} that is higher along the frontiers of the graph, where the agent has less experience. To turn this into an expansion probability, we ask: if the agent executes a novelty-maximizing policy π_{novelty} starting from each graph node, how much intrinsic reward can it expect to accumulate? Fortunately, this concept is well captured by the exploration value function V_{novelty}. But, solely using novelty will result in graph expansion in all possible directions; to forego exhaustive coverage and prioritize high reward regions, we add the intrinsic and extrinsic rewards together while learning V_{novelty}. Concretely, we define the utility of expanding a node n as $U(n) = \mathbb{E}_{s\sim n}\Big[V_{\text{novelty}}(s)\Big] = \mathbb{E}_{s\sim n}\Big[\sum_{t=1}^{\infty} R(s_t, a_t) + b \cdot r_{\text{novelty}}(s_t) \mid s_0 = s\Big]$; we use $b = 0.01$ in all our experiments.

Finally, we restrict the choice of expansion node to the set of nodes that are reachable from the current state, i.e, the descendants of the nodes that map to the current state of the agent s_t. Therefore, we define the probability of expanding a node n as follows:

$$n_{\text{expansion}} \sim \mathbb{P}_n = \frac{U(n)}{\sum_{n' \in \mathcal{D}(s_t)} U(n')} = \frac{\mathbb{E}_{s \sim n}\Big[V_{\text{novelty}}(s)\Big]}{\sum_{n' \in \mathcal{D}(s_t)} \mathbb{E}_{s' \sim n'}\Big[V_{\text{novelty}}(s')\Big]}, \forall n \in \mathcal{D}(s_t). \tag{7.4}$$

7.2.2.2 Estimating Edge Probabilities

Every edge in the skill graph has a weight $p_{n_i \to n_j}$, which represents the probability with which an option o_{n_j} initiating from states in node n_i will successfully terminate in node n_j. As pointed out earlier, this is equivalent to the expected initiation probability of option o_{n_j}, where the expectation is taken with respect to states inside the node n_i. Following Bagaria et al. (2023), we interpret the agent's goal-conditioned value function (UVFA) (Schaul et al., 2015) as an initiation probability:

$$p_{n_i \to n_j} = \mathbb{E}_{s \sim n_i}\Big[\mathcal{I}_{o_j}(s)\Big] \approx \mathbb{E}_{s \sim n_i}\Big[V_{n_j}(s)\Big] := v_{n_i \to n_j}. \tag{7.5}$$

It is possible to infer the edge probabilities from the UVFA because the goal-conditioned reward function that is used to train it is $\mathcal{R}_o(s) = \beta_o(s)$, i.e., it is $+1$ for states inside the option's subgoal region and 0 otherwise. However, naively using the UVFA prediction as edge probability can hinder learning because:

1. $v_{n_i \to n_j}$ for an edge can be low if traversing it is actually unlikely in a single option execution (even under an optimal policy) *or* because the agent does not yet have enough data, and,
2. when we add new nodes to the graph, the probability of reaching them will initially be low. To increase these edge probabilities, the agent must continually practice the options that target them, but this is unlikely to happen owing to their low starting

probabilities.

Therefore, to address these issues, we are *optimistic* about the value of the edge probabilities:

$$p^{+}_{n_i \to n_j} := v_{n_i \to n_j} + \mathcal{U}(n_i, n_j) \approx v_{n_i \to n_j} + \frac{c}{\sqrt{N[n_i, n_j] + 1}}, \tag{7.6}$$

where $\mathcal{U}(n_i, n_j)$ represents the uncertainty about our value prediction $v_{n_i \to n_j}$, which we obtain using $N[n_i, n_j]$ (Brafman and Tennenholtz, 2002; Strehl and Littman, 2008), the number of times the agent has executed option o_{n_j} starting from states in node n_i ($c \in [0, 1]$ is a hyperparameter).[1] When $N[n_i, n_j]$ is low, the agent is optimistic about traversing that edge, and so it is encouraged to practice the corresponding option policy. But, as the agent attempts to traverse edge $e_{n_i \to n_j}$, the count $N[n_i, n_j]$ increases, and the optimistic bias decays; eventually, the edge probability approaches the goal-conditioned value, which we expect to become more accurate with more data. By incorporating an optimistic bias via the edge probabilities, we ensure that the policies computed by the planner achieve a form of **exploration within the graph**, which prioritizes paths that reach the expansion node while improving newly added skills.

7.2.2.3 Deciding when there is an edge between two nodes

Using Equation 7.6, we can determine the probability of traversing an edge in a single option execution. Naïvely, this would result in a dense, fully-connected graph, which would be problematic: suppose there is an existing node n_1 and we just added a new node n_2 that is far away (even under an optimal policy) from n_1. The agent's initial desire to connect them would be high (because $N[n_1, n_2]$ will be low). As a result, the agent will spend a lot of time realizing that n_1 and n_2 should actually have a low probability of being directly connected to one another. Not only is this sample-inefficient, it also results

---

[1] Since the counts are over discrete states of the AMDP, we simply maintain tabular counts in a $|\bar{S}| \times |\bar{S}|$ matrix.

in graph construction *slowing down over time*: for every new node added to the graph, the agent must determine how the new node relates to every other existing node. To mitigate this issue, we use the following heuristics to determine when two nodes n_1 and n_2 should have an edge connecting them:

- **Observed multi-step transition.** When we observe a transition between two nodes $n_1 \to n_2$ within H steps (the maximum horizon of an option), we connect them with an edge.
- **High goal-conditioned value.** We connect nodes $n_1 \to n_2$ with an edge when the goal-conditioned value between them, $\mathbb{E}_{s \sim n_1}\Big[V_{n_2}(s)\Big]$, is high enough. To ensure that the resulting model is amenable to planning, we add edges only when we are *highly confident* that execution of one option will allow for the execution of another (Konidaris et al., 2018). We instantiate this principle via an adaptive threshold that is high to begin with, and tapers to a fixed hyperparameter $\tau \in [0, 1]$ when more experience is gained:

  $$e_{n_1 \to n_2} = \mathbb{I}\left[\mathbb{E}_{s \sim n_1}\Big[V_{n_2}(s)\Big] - \frac{c}{\sqrt{\min\{N[n_1], N[n_2]\}}} > \tau\right], \tag{7.7}$$

  where $\mathbb{I}$ is the indicator function and $N[n]$ is the number of times the agent has visited node n. The inverse square root term acts as an uncertainty penalty that diminishes as more data is collected.

7.2.2.4 Constructing the Abstract MDP

After the agent identifies the expansion node $n_{\text{expansion}}$, it needs to compute a plan that will guide it from its current state s_t to $n_{\text{expansion}}$. Our graph has two properties that will inform an appropriate choice of planning algorithm:

1. **Probabilistic transitions**. The skill graph has probabilistic edges because (a) the environment's transition function and the agent's option policies are stochastic,

and (b) the option policies are being learned online, causing the abstract transition function to also appear stochastic and nonstationary (Nachum et al., 2018; Levy et al., 2019; Bagaria et al., 2021a). We represent these dynamics with a transition matrix, $\bar{T}$, where each entry $\bar{T}_{ij}$ is the probability that executing the option targeting node n_j from node n_i successfully leads to node n_j. When option execution is unsuccessful, we assume the agent "falls off" the graph, modeling it by assigning the remaining probability $(1 - \bar{T}_{ij})$ to an absorbing failure state φ.

2. **Incorporating extrinsic reward.** Given two paths from s_t that both reach $n_{\text{expansion}}$, we would prefer one that collects extrinsic rewards along the way. This allows us to form a reward-respecting abstract model (Sutton et al., 2024), and prioritizes skill refinement in parts of the graph that are likely to result in higher reward. To incorporate this, we define a reward matrix, $\bar{R}$, where each entry $\bar{R}_{ij}$ is the sum of extrinsic rewards collected during the execution of the option and an intrinsic reward that is high at the expansion node. Formally, we write: $\bar{R}_{ij} = \kappa \bar{R}^{\text{ext}}_{ij} + \bar{R}^{\text{int}}_{j}$, where $\kappa \in \mathbb{R}$ scales the extrinsic reward, and $\bar{R}^{\text{int}}_{j}$ equals 1 if node n_j is the expansion node, and 0 otherwise.

We also define an abstract discount factor, $\bar{\gamma}$, which reflects the probability of staying within the graph during an option execution. Putting these elements together, our AMDP is defined as: $\bar{M} = (\bar{S}, \bar{A}, \bar{R}, \bar{T}, \bar{\gamma})$, where $\bar{S}$ and $\bar{A}$ represent the set of nodes in the skill graph. Planning on this abstract MDP is a Stochastic Shortest Path (SSP) problem (Bertsekas and Tsitsiklis, 1991). We solve this SSP using value iteration (Bellman, 1966), which yields a deterministic abstract policy $\bar{\pi}$. This abstract policy maps each node (abstract state) to an option, guiding the agent toward the expansion node while balancing the collection of extrinsic rewards and exploration of uncertain edges. An illustrative example of this AMDP construction is provided in Figure 7.2.

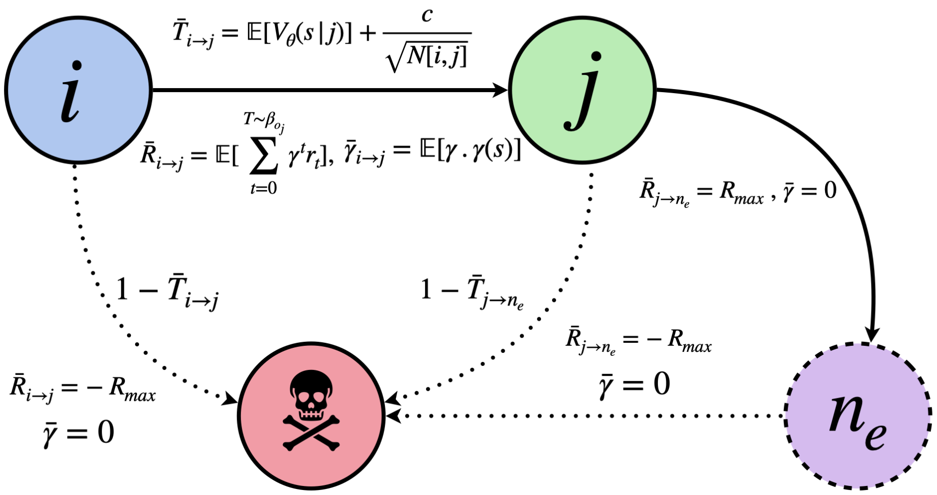

i
j
n_e
$\bar{T}_{i\to j} = \mathbb{E}[V_\theta(s|j)] + \frac{c}{\sqrt{N[i,j]}}$
$\bar{R}_{i\to j} = \mathbb{E}[\sum_{t=0}^{T\sim\beta_{o_j}} \gamma^t r_t]$, $\bar{\gamma}_{i\to j} = \mathbb{E}[\gamma \cdot \gamma(s)]$
$\bar{R}_{j\to n_e} = R_{max}$, $\bar{\gamma} = 0$
$1 - \bar{T}_{i\to j}$
$1 - \bar{T}_{j\to n_e}$
$\bar{R}_{j\to n_e} = -R_{max}$
$\bar{\gamma} = 0$
$\bar{R}_{i\to j} = -R_{max}$
$\bar{\gamma} = 0$

Figure 7.2: Summary illustration of a toy abstract MDP (AMDP) with 3 nodes. i and j are two ordinary nodes, n_e is the expansion node and the skull node shows a fictitious node that represents "falling off the graph". The transition probabilities $\bar{T}_{i\to j}$ are obtained using the UVFA V_θ and edge-traversal counts $N[i,j]$; the reward $\bar{R}_{i\to j}$ and discount models $\bar{\gamma}$ are obtained via monte carlo estimation from option executions.

7.2.2.5 Navigating to the Expansion Node

The abstract policy $\bar{\pi}$ tells the agent which options to execute to reach the expansion node. While following $\bar{\pi}$, if the agent finds itself in a state that is not in the graph ($s_t \in \varphi$), it targets the closest node in the graph: $o_t = \arg\max_o V_{\beta_o}(s_t), \forall o \in \mathcal{O}$. Each option execution corresponds to rolling out the corresponding option policy $\pi_o(s)$, which is trained using recurrent deep Q-learning (Kapturowski et al., 2019) to maximize the pseudo reward function $R_o(s) = \beta_o(s)$. In practice, all option policies are parameterized using the same goal-conditioned policy $\pi(s|g;\theta)$ (Schaul et al., 2015) and each option conditions the policy with goal states sampled from its own termination region (Bagaria et al., 2021a). To improve the sample-efficiency of learning effective option policies, we use hindsight experience replay (Andrychowicz et al., 2017).

7.2.2.6 Generating Behavior Outside the Graph

After reaching the expansion node, the IM-DSG agent attempts to extend the graph. It does so by executing the low-level exploration policy π_{CFN}, which is trained to maximize the weighted sum of intrinsic and extrinsic rewards:

$$R_{\text{CFN}}(s, a) = R(s, a) + b \cdot r_{\text{novelty}}(s),$$

where $r_{\text{novelty}}(s) = 1/\sqrt{N_\phi(s)}$, $b \in [0, 1]$ controls the exploration-exploitation trade-off (we use $b = 0.01$) and $N_\phi(s)$ is the CFN visitation count prediction for state s. We use recurrent deep Q-learning (Kapturowski et al., 2019) to learn V_{novelty} and π_{novelty}. The result of this policy execution is a trajectory τ_{novelty}, which is likely to move outward, toward regions where the agent's abstract model is more uncertain; this is a way of doing exploration *outside* the graph.

7.2.2.7 Adding a New Node to the Graph

Having generated a low-level exploration trajectory τ_{novelty} in the frontier of the graph, the agent must now identify a new node to add to the graph. Again, we take inspiration from intrinsic motivation methods (Şimşek and Barto, 2004): the agent identifies the state in τ_{novelty} with the highest predicted novelty; if the novelty of that state is higher than the running mean of the novelty-based rewards, then we flag it as a new target and add it to the skill graph:

$$s_{\text{target}} = \underset{s \in \tau_{\text{novelty}}}{\arg\max} \Big[r_{\text{novelty}}(s) \Big] \text{ if } \max_{s} \Big[r_{\text{novelty}}(s) \Big] > \mu_t + k\sigma_t, \text{ else } \varphi, \tag{7.8}$$

where μ_t is the running mean and σ_t is the running standard deviation of r^{novelty}, computed incrementally over the lifetime of the agent (**?**); $k \in \mathbb{R}^+$ is a hyperpameter (we use $k = 1$). Equation 7.8 identifies a target state, which we need to convert to a termination classifier so that it can serve as a node in the skill graph. We could use domain knowledge (for example, the "cell" representations of Ecoffet et al. (2021)), pixel-wise equality (Veeriah et al., 2018), or a temporal distance based classifier (Savinov et al., 2018); we opt for simplicity and use an off-the-shelf template matching algorithm, which is a simple call to

OpenCV's `matchTemplate`(s, g) (Bradski, 2000):

$$n_{\text{new}} = \beta_{o_{\text{new}}}(s) = \texttt{matchTemplate}(s, s_{\text{target}}). \tag{7.9}$$

7.2.2.8 Summary of Graph Discovery Algorithm

Algorithm 1 Intrinsic Motivation Directed Skill Graph (IM-DSG)

```
Initialize:
Initialize CFN to learn pseudocounts $N_\phi$, value function $V_{\text{novelty}}$, and policy $\pi_{\text{novelty}}$.
Initialize goal-conditioned R2D2 to learn value function $V_g$ and policy $\pi_g$.
Initialize graph $\mathcal{G}$ with null node $\{\varphi\}$ and no edges.
while True do
    Sample expansion node using Equation 7.4.
    Construct AMDP using procedure described in Section 7.2.2.4.
    Solve AMDP using value iteration to get abstract policy $\bar{\pi}$.
    Follow abstract policy until agent reaches expansion node or episode terminates.
    if agent reaches expansion node then
        Roll out exploration policy $\pi_{\text{novelty}}$ to get trajectory $\tau$.
    end if
    Potentially add a new node from $\tau$ to the graph using Equation 7.8.
    Update $V_g$ and $\pi_g$ using Hindsight Experience Replay (HER).
    Update $\pi_{\text{novelty}}$ to maximize the sum of intrinsic and extrinsic rewards.
    Update the edge traversal $N[i, j]$ and node visitation counts $N[i]$.
    Update the edge probabilities in $\mathcal{G}$ using $V_g$ (Sections 7.2.2.2 and 7.2.2.3).
end while
```

Algorithm 1 summarizes IM-DSG. The agent maintains two value functions V_g, V_{novelty} and two policies $\pi_g, \pi_{\text{novelty}}$; R2D2 (Kapturowski et al., 2019) is used to learn both sets of policies and value functions. The agent first chooses an expansion node using Equation 7.4. Which option is executed is then determined by solving the resulting Abstract MDP using value iteration. New nodes are added to the graph by executing π_{novelty} from the expansion node. Finally, graph edges are updated based on V_g and edge traversal counts $N[i, j]$. The choice of expansion node and subsequent low-level exploration encourage the skill graph to expand along directions that jointly maximize novelty and extrinsic reward.

7.3 Experiments

We evaluate the IM-DSG agent on five challenging exploration problems, all of which have image-based observations and discrete actions (details in Section 7.3.1). We chose the MiniGrid (Chevalier-Boisvert et al., 2023) environments because they were identified as being challenging for existing RL algorithms by the survey of Colas et al. (2022). We chose Sokoban (Schrader, 2018) because the survey of Hamrick et al. (2020) identified it as being a problem in which both learning and planning are important for good performance. Finally, we chose VisualTaxi and VisualPinball because non-image versions of these domains have been used extensively to evaluate state abstraction and option discovery algorithms in isolation (for example, Lo et al., 2024; Konidaris and Barto, 2009b; Dietterich, 2000).

7.3.1 Domain Descriptions

1. MiniGrid-DoorKey: To reach the goal, the agent must first pick up a key and then use that key to unlock a door. There is no intermediate reward for picking up the key, only a sparse terminating reward for reaching the goal. Each episode lasts a maximum of 200 steps.

2. MiniGrid-KeyCorridor: This domain also has a key and a locked door, but additionally has other doors that can be open and closed. The goal is to pick up the purple ball that is placed in locked room (center-right room in Figure 7.3(c)). Each episode lasts a maximum of 1000 steps.

3. Sokoban: In this classic puzzle, the agent must place 3 boxes into their target locations (red border squares in Figure 7.3). The environment provides intermediate rewards for putting each box into its place and a terminating reward for correctly placing the final box. Each episode lasts a maximum of 200 steps.

4. VisualTaxi: this is an image-based version of the classic Taxi problem (Dietterich,

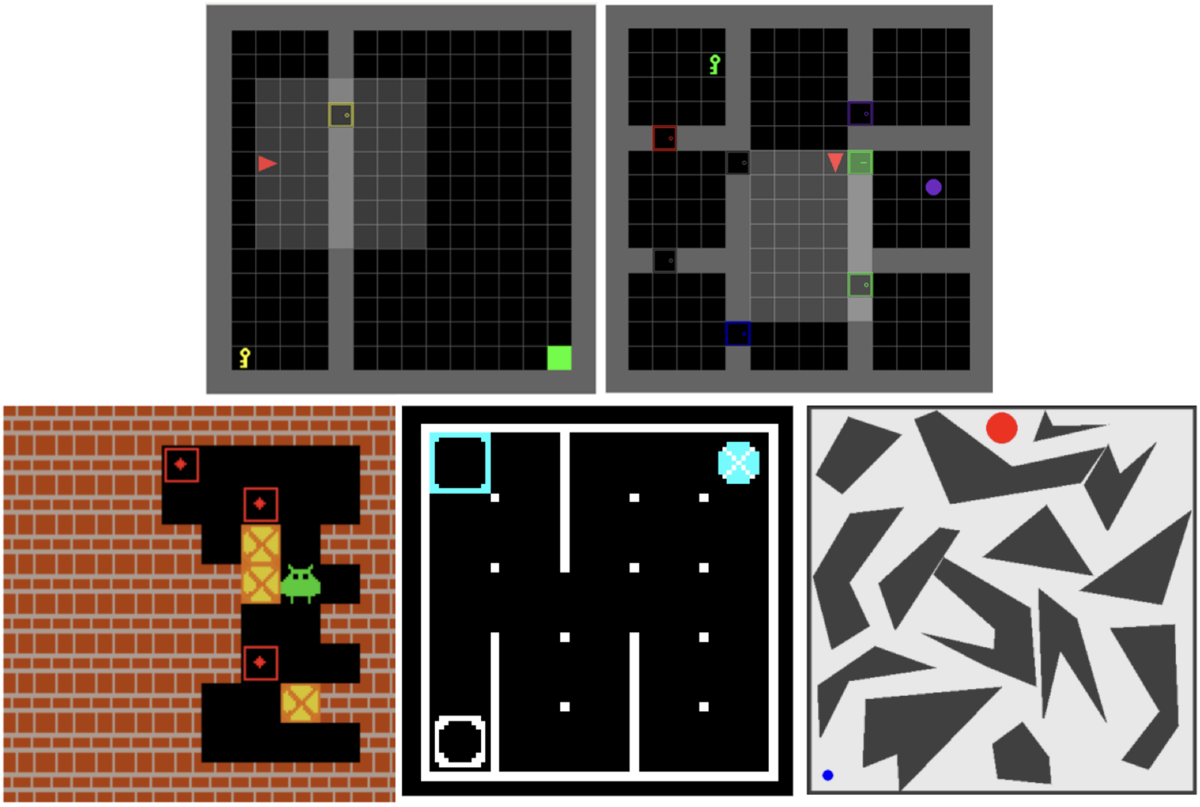


Figure 7.3: Domains used to test the IM-DSG agent. From top-left in left to right order: MiniGrid-DoorKey, MiniGrid-KeyCorridor, Sokoban, VisualTaxi, and VisualPinball.

2000). A passenger awaits in one of four depots and must be dropped off at a destination depot. Successful completion of the full task yeilds a sparse terminating reward of +1. Each episode lasts a maximum of 50 steps.

5. VisualPinball: image-based version of the under-actuated Pinball domain (Konidaris and Barto, 2009b; Bacon et al., 2017) in which the agent provides acceleration to the ball in one of 4 directions (or no-op), causing the velocity of the ball to change over time. We pick the hardest configuration from Konidaris and Barto (2009b) during training, and sample goal locations at random during testing. Test-time goals are communicated to the agent as images of the pinball in the desired location. Each episode lasts a maximum of 1000 steps.

7.3.2 Qualitative Evaluation

To build intuition, we first qualitatively evaluate the IM-DSG agent. Specifically, we want to understand what the skill-graph looks like, what parts of it the agent uses most frequently, and whether the skill graph aids more effective exploration.

7.3.2.1 Visualizing the Skill Graph in MiniGrid

Figure 7.4 is a visualization of the skill graph at the end of training in MiniGrid-KeyCorridor. To visualize each node in the graph, we randomly sample a state in which that termination condition was triggered; only the location and inventory is visualized here for simplicity. Each node is colored based on its probability of serving as the expansion node. Furthermore, the transparency α_n of a node is its average abstract value:

$$\alpha_n = \max \left\{ \mathbb{E}_{n_e} \left[\bar{V}_{n_e}(n) \right], 0.05 \right\},$$

where the expectation is over the node expansion probability distribution $\mathbb{P}_n$ (Equation 7.4), $\bar{V}_g(n)$ is the abstract value of node n when targeting node g (obtained via Value Iteration on the AMDP; Section 7.2.2.4). The transparency of an edge is the average transparency of the nodes that it connects.

Understanding Figure 7.4. Nodes in the top-right room have a higher expansion probability. This is because (a) navigating to the top-right room is novel as it requires unlocking the red door with the red key, and (b) once the agent enters the top-right room, it can solve the remainder of the task easily. These two facts result in high V_{novelty} and hence high expansion probabilities, which is why the graph in the bottom-left and top-right rooms have high abstract value. Nodes in the central corridor also have high abstract value because the agent must navigate through them for the majority of its target expansion nodes. This demonstrates that the agent focuses on parts of the graph that are either important for expansion by themselves, or those subgraphs that serve as *hubs*, i.e.,

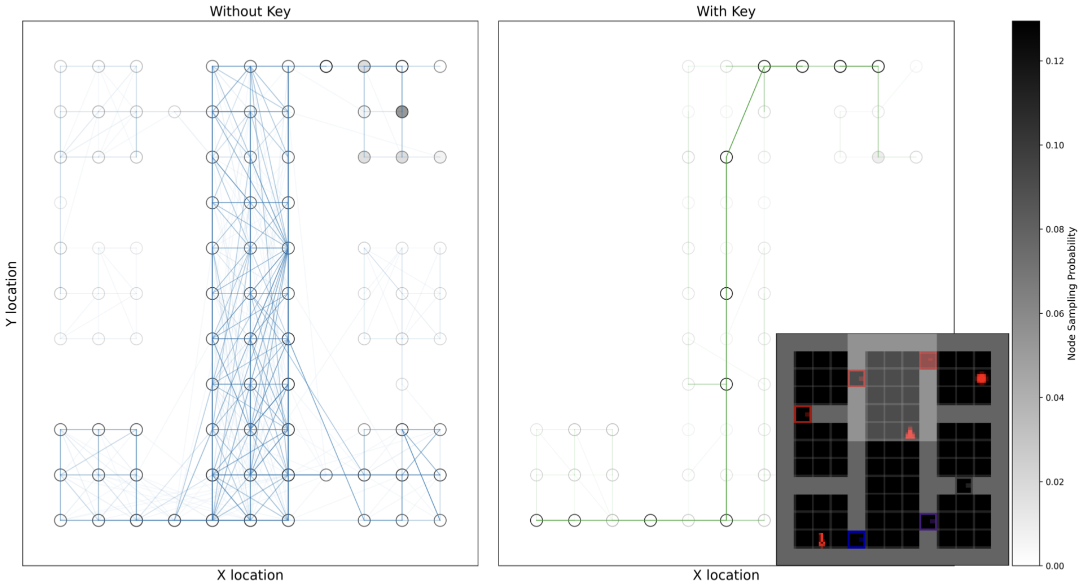


Figure 7.4: **Discovered skill graph in MiniGrid-KeyCorridor**. The domain is shown in the bottom-right inset: it involves picking up a key in the bottom-left room, opening a locked door and then picking up a ball in the top-right room. The left subplot shows the subgraph where the agent does *not* have a key in its possession. The nodes and edges of the graph are shaded based on their expected abstract value $\mathbb{E}[\bar{V}(n)]$. Although we visualize the location and inventory, the agent itself only observes images. The graphs show that, when the agent lacks the key, it aims to explore down the corridor and in the rooms at the bottom of the maze, but when it does have the key, interesting options are on the path to the goal.

they lead to other sub-graphs that are important for expansion (Şimşek et al., 2005; Shah and Srivastava, 2024).

7.3.2.2 Exploration in Sokoban

Figure 7.5 shows IM-DSG's high-level policy when it encounters the task-goal for the first time: the expansion node is chosen to be a state that corresponds to placing two out of the three boxes in their correct locations. The agent plans with its learned options to reach this state. Subsequent novelty-driven exploration is more targeted as it only seeks to explore those states in which most of the task has already been solved, increasing the probability that random exploration would find the proverbial “needle in a haystack”.

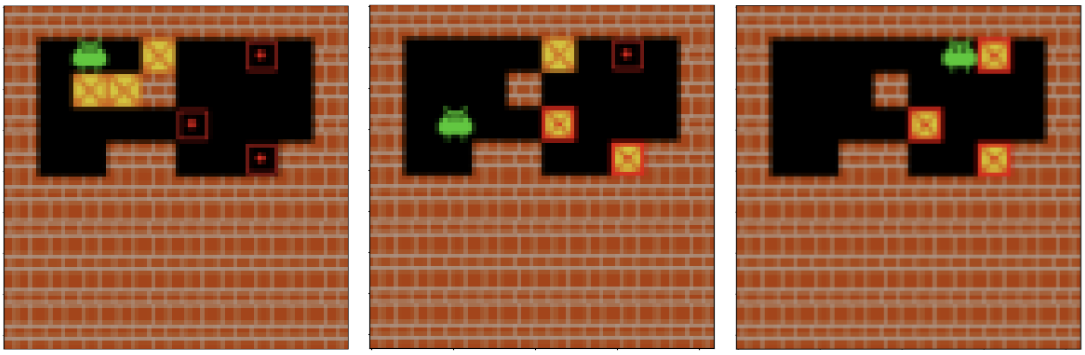


Figure 7.5: Example of hierarchical policy execution in SOKOBAN: *(left)* start state in which *no* box (boxes shown in yellow) is in its desired location (target locations are shown in red). *(Middle)* selected expansion node which has 2 out of 3 boxes placed in their desired locations; the agent uses planning to get to this state. *(Right)* After reaching the expansion node, the agent explores locally using π_{novelty}, which reaches the task goal.

7.3.3 Quantitative Evaluation

First, we seek to understand whether the IM-DSG agent can solve challenging exploration problems in a more sample-efficient manner than our baselines. Second, even though the graph construction is prioritized along directions where the agent expects to gather extrinsic reward, we want to understand whether the graph can still be used in a multi-task context to solve goal-reaching problems that were not encountered during training.

7.3.3.1 Sample Efficient Exploration in Single Task MDPs

We compare the IM-DSG agent to the following baselines:

- **Flat model-free RL.** R2D2 is a distributed variant of DQN that uses a recurrent neural network to represent the value function (Kapturowski et al., 2019). Since the IM-DSG agent uses R2D2 to learn policies, we include it as our model-free baseline.
- **Novelty driven exploration.** We use CFN to explore outside the graph, and it is a state-of-the-art technique for novelty-driven RL; so, we include it as our exploration baseline.

Figure 7.6 shows our quantitative results: we find that the IM-DSG agent consistently

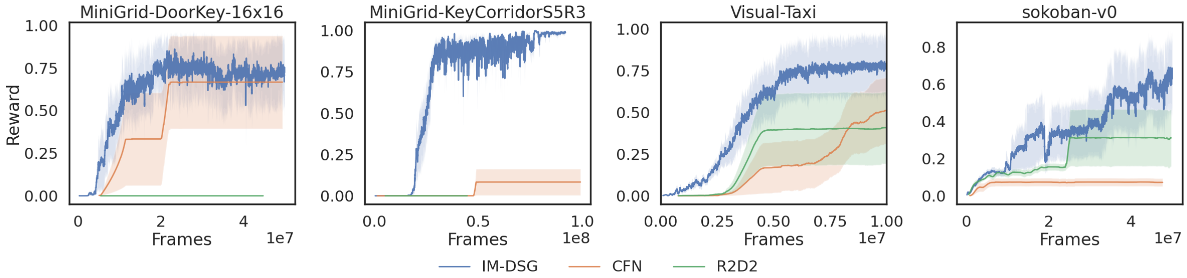


Figure 7.6: Comparing the performance of our proposed algorithm, IM-DSG (blue), novelty-based CFN (orange) and vanilla model-free R2D2 (green). Each curve represents the mean reward and the shaded area represents the standard deviation over 5 random seeds.

outperforms R2D2 and CFN in our domains. This suggests that although novelty-maximizing exploration is in principle sufficient to solve these problems, the discovery and use of the model for planning leads to a more sample-efficient agent in these domains.

Model-based planning ablation. Figure 7.7 *(left)* shows that when we disable planning, the IM-DSG agent is unable to solve the MiniGrid-KeyCorridor problem. This indicates that high-level planning with the learned skill graph is essential–simply combining CFN with HER is not enough to solve this task in the allotted training budget. This ablation also serves a rough comparison to algorithms like Skew-Fit (Pong et al., 2019) and Omega (Pitis et al., 2020c) that combine HER with novelty-based exploration, but do not do model-based planning.

7.3.3.2 Generalization to New Goals in VisualPinball

In this section, we test whether the skill graph discovered during exploration can be used to solve other, unseen tasks at test-time. In particular, we train our agent to solve the VisualPinball problem, which requires guiding the ball to the big red circle at the bottom of the maze. After training, we present the agent with 10k randomly sampled held-out goals and test its ability to reach these goals. We compare our agent against Hindsight Experience Replay (HER; Andrychowicz et al., 2017), which in principle can also generalize to unseen goals at test-time.

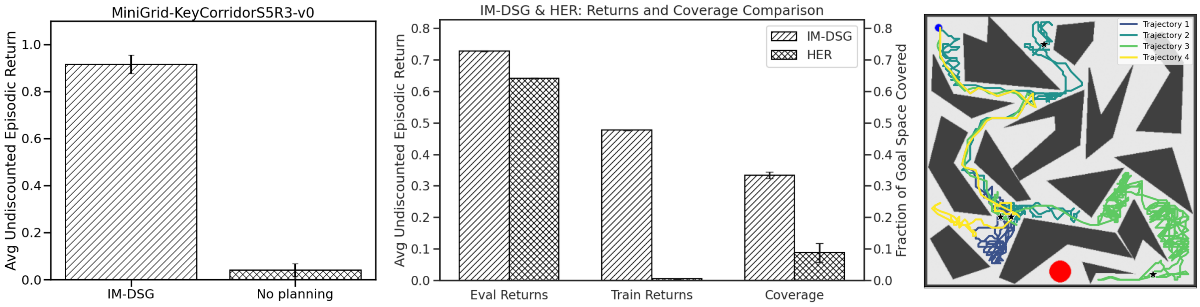


Figure 7.7: **Ablation and graph usefulness in a multi-task context.** *(left)* Ablating the use of planning in IM-DSG in MiniGrid-KeyCorridor: bars represent the average per-episode reward during training. *(middle)* Evaluation performance in VisualPinball on 10k randomly sampled goal states not used during training: comparison against HER in terms of test-time rewards, train-time rewards, and coverage, which is the fraction of the maze explored during training. All error bars represent standard deviation over 5 random seeds. *(right)* Solution trajectories found by the IM-DSG agent for 4 randomly sampled test-time goals (shown using black stars); the blue dot in the top-left of the maze is the start state, the big red ball in the bottom is the train-time goal. Note that, unlike in the original, this version of pinball uses images as observations.

Both the IM-DSG agent and the Hindsight Experience Replay (HER) agent get the same training budget of 50M frames. During evaluation, both algorithms are tested on the same set of 10k goals and are given a budget of 10M frames to achieve them. Each episode lasts for at most for 1k steps. Error bars are computed over 5 training runs of the respective agents.

To compute the coverage metric shown in Figure 7.7 *(right)*, we first need to compute the maximum possible coverage in the maze. This is done by first discretizing the (x, y) plane into 100×100 squares, and then subtracting the approximate area occupied by the obstacles in the maze. To compute the coverage achieved by the two agents, we consider what fraction of the free space is occupied by all the states visited by the two agents.

Test-time modifications for VisualPinball. During training, IM-DSG picks the expansion node based on V_{novelty}. At test-time, this would not be a good strategy because there is no reason that V_{novelty} would guide the agent to the test-time goal. So instead, we pick the expansion node to be the one that is closest to the test-time goal g_{test}:

$$n^{\text{test}}_{\text{expansion}} = \arg\max_{n} \mathbb{E}_{s\sim n}\Big[V_{g_{\text{test}}}(s)\Big] \tag{7.10}$$

Once the agent reaches $n^{\text{test}}_{\text{expansion}}$, it then executes its goal-conditioned policy $\pi(s, g_{\text{test}})$ to reach the test-time goal g_{test}.

Figure 7.7 shows that IM-DSG outperforms HER at test-time. To understand why, we compare their train-time performances and find that HER is unable to explore the maze as extensively as IM-DSG, as a result of which, it is unable to reach goals that are far away from the start-state at test-time.

7.4 Discussion and Conclusion

We introduced an algorithm that builds a graph abstraction of large MDPs with image-based observations. Nodes of this graph correspond to option termination regions, edges represent option policies, and edge probabilities are determined using option initiation functions. The skill graph expands towards the frontier of the agent's experience without assuming access to state-sampling or a distance metric; the acquired graph enables planning in increasingly large portions of the state-space.

Our method suffers from at least the following limitations:

- Each option drives the value of all state variables to a certain range of values. In more complex environments, it may be unnecessary, or even impossible, to control all state variables at once.
- We build a **dense** graph abstraction of the environment. This was not an issue in the environments we considered in this paper, but scaling to larger environments could be expensive in terms of memory usage. This might also necessitate more sophisticated asynchronous dynamic programming methods such as prioritized sweeping (**?**).

- We do not use deep skill chaining (Chapter 3), which could accelerate graph expansion by permitting the agent to reach the expansion node with greater reliability.
- After reaching the expansion node, the IM-DSG agent executes the low-level exploration policy (CFN) until the end of the episode; more fine-grained control over when to enter and exit low-level exploration (Pislar et al., 2022) would generalize our algorithm to continuing problems without episode resets.

Despite these shortcomings, IM-DSG takes a small step towards the eventual goal of building RL agents that continually discover their own state and action abstractions (Gershman, 2017; Konidaris, 2019), and use them for effective exploration (Gopnik, 2020) and planning (Sutton et al., 2024).

CHAPTER 8

Scaling Goal-based Exploration via Pruning Proto-goals

Exploration is widely recognised as a core challenge in RL. It is most acutely felt when scaling to vast domains, where classical novelty-seeking methods are insufficient (Taïga et al., 2019) because there are simply too many things to observe, do, and learn about; and the agent's lifetime is far too short to approach exhaustive coverage (Sutton et al., 2022). Abstraction can overcome this issue (Gershman, 2017; Konidaris, 2019): by learning about goal-directed, purposeful behaviours (and how to combine them), the RL agent can ignore irrelevant details, and effectively traverse the state space. Goal-conditioned RL is one natural formalism of abstraction, and especially appealing when the agent can learn to generalise across goals (Schaul et al., 2015).

The effectiveness of goal-conditioned agents directly depends on the size and quality of the goal space (Section 8.2). If it is too large, such as treating all encountered states as goals (Andrychowicz et al., 2017), most of the abstraction benefits vanish. On the other extreme, hand-crafting a small number of useful goals (Barreto et al., 2019a) limits the generality of the method. To help find the "sweet spot", I propose that the agent adaptively expand or refine the goal space based on experience, allowing for a more

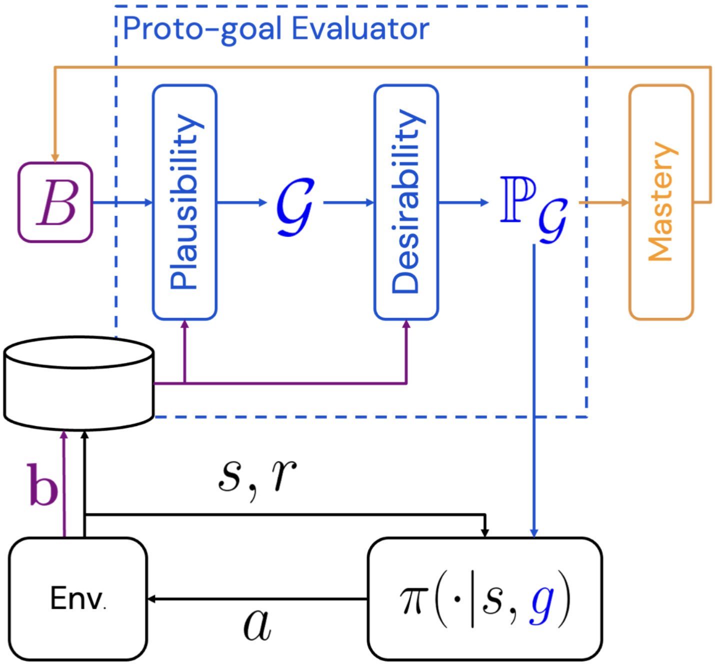


Figure 8.1: **Proto-goal RL**: a goal-conditioned RL agent's policy π acts with goals g obtained from its Proto-goal Evaluator (PGE, blue). The PGE refines a cheaply defined *proto-goal space* (B, violet) into a smaller set of plausible goals $\mathcal{G}$, using observed transition data (s, a, r, s') that includes information about encountered proto-goals ($\mathbf{b}$). It further endows $\mathcal{G}$ with a distribution $\mathbb{P}_{\mathcal{G}}$, based on goal desirability, from which g is then sampled. When goals from $\mathbb{P}_{\mathcal{G}}$ are mastered (orange), they are recombined to create new proto-goals that are added to the proto-goal space.

autonomous agent that can be both general and scalable.

Taking a step towards this ultimate aim, I propose a framework with two elements. The first element of this framework is what I call a *proto-goal* space (Section 8.2), which is a large space of candidate goals that can be cheaply designed to be meaningful for the domain at hand, e.g., by pointing out the most salient part of an observation using domain knowledge (Chentanez et al., 2005). What makes defining a proto-goal space much easier than defining a goal space is its leniency: it can remain (combinatorially) large and unrefined, with many uninteresting or useless proto-goals. The second element of this framework is an adaptive function mapping the proto-goal space to a compact set of useful goals, called a *Proto-goal Evaluator* (PGE, Section 8.3). The PGE may employ multiple criteria of usefulness, such as controllability, novelty, reachability, learning progress, or reward-relevance. Finally we address pragmatic concerns on how to integrate these elements into a large-scale goal-conditioned RL agent (Section 8.4), and show it can produce a qualitative leap in performance in otherwise intractable exploration domains (Section 8.5).

8.1 Background and Related Work

We consider problems modeled as Markov Decision Processes (MDPs) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R}$ is the reward function, $\mathcal{T}$ is the transition function and γ is the discount factor. The aim of the agent is to learn a policy that maximises the sum of expected rewards (Sutton and Barto, 2018).

Exploration in RL. Many RL systems use myopic strategies for exploration (e.g., ϵ-greedy, softmax, action-noise (Lillicrap et al., 2015) and parameter noise (Fortunato et al., 2018; Plappert et al., 2018)). Among those that address *deep* exploration (Osband et al., 2016a), the majority of research (Taïga et al., 2019) has focused on count-based exploration (Strehl and Littman, 2008; Bellemare et al., 2016), minimizing model prediction error (Pathak et al., 2017; Burda et al., 2019a,b), or picking actions to reduce uncertainty (Osband et al., 2016b, 2018) over the state space. These strategies try to eventually learn about *all* states (Ecoffet et al., 2021), which might not be a scalable strategy when the world is a lot bigger than the agent (Javed and Sutton, 2024). We build on the relatively under-studied family of exploration methods that maximize the agent's *learning progress* (Schmidhuber, 1991; Kaplan and Oudeyer, 2004; Colas et al., 2022).

Goal-conditioned RL. When the space of GVF cumulants is limited to control, GVFs reduce to goal-conditioned value functions that are often parameterized using Universal Value Function Approximators (UVFAs) (Schaul et al., 2015). Hindsight Experience Replay (HER) is a popular way of learning UVFAs in a sample-efficient way (Andrychowicz et al., 2017): in addition to learning about the goal that the agent was pursuing, it additionally learns about a goal that was achieved in a trajectory in an off-policy way. The two most common approaches to goal-conditioned RL are to assume that a set of goals is given, or to treat all observations as potential goals (Liu et al., 2022) and then learn a policy that can reach *any* state. In large environments, the latter methods often over-explore (Pong et al., 2019; Pitis et al., 2020b) or suffer from interference between goals (Schaul et al., 2019).

Discovery of goals and options. Several objectives have been proposed for goal/option discovery: reward relevance (Bacon et al., 2017; Veeriah et al., 2021), composability (Konidaris and Barto, 2009b; Bagaria and Konidaris, 2020a), diversity (Eysenbach et al., 2019a; Campos Camúñez et al., 2020), empowerment (Mohamed and Jimenez Rezende, 2015), coverage (Bagaria et al., 2021b; Machado et al., 2017), etc. These heuristic objectives measure *desirability*, but they must be paired with *plausibility* metrics like controllability and reachability to discover meaningful goals in large goal spaces. The IMGEP framework (Forestier et al., 2022) also does skill-acquisition based on competence progress, but they assume more structure in the goal space (e.g., Euclidean measure, objects), and use evolution strategies to represent policies instead of RL. STOMP (Sutton et al., 2024) learns feature attainment options, which are similar to proto-goal achieving policies; but unlike STOMP, we maintain different representations for states and goals. Furthermore, they do not provide an explicit way to prune large feature/goal spaces, nor do they construct new features/goals out of existing ones.

8.2 Goals and Proto-goals

A *goal* is anything that an agent can pursue and attain through its behaviour. Goals are well formalised with a scalar cumulant $c_g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ and a continuation function $\gamma_g : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, as proposed in the general value function (GVF) framework (Sutton et al., 2011). Here, we consider the subclass of *attainment* goals g, or “endgoals”, which imply a binary reward that is paired with termination. In other words a transition has either ($c_g = 0, \gamma_g > 0$) or ($c_g = 1, \gamma_g = 0$), i.e., only terminal transitions are rewarding. The corresponding goal-optimal value functions satisfy:

$$Q_g^*(s, a) = \mathbb{E}_{s'}\left[c_g(s, a, s') + \gamma_g(s, a, s') \max_{a'} Q_g^*(s', a')\right],$$

with corresponding greedy policy $\pi_g^* := \arg\max_a Q_g^*(s, a)$.

Proto-goals are candidate goals. Since attainment goals can easily be derived from any binary function, we formally define a proto-goal to be a binary function of a transition $b_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \{0, 1\}$. We assume that, for a given domain, a set B of such proto-goals can be queried. Proto-goals differ from goals in two ways. First, to fully specify a goal, a proto-goal must be paired with a time-scale constant $\gamma \in [0, 1]$ (a discount), which defines the horizon over which g should be achieved. The pair (b_i, γ) then define the goal's cumulant $c_g(s, a, s') := b_i(s, a, s')$ and continuation function $\gamma_g(s, a, s') := \gamma(1 - b_i(s, a, s'))$. Second, less formally, the space of proto-goals B is vastly larger than any reasonable set goals $\mathcal{G}$ that could be useful to an RL agent. Hence the need for the Proto-goal evaluator (Section 8.3) to convert one space into the other.

8.2.1 Example Proto-goal Spaces

A proto-goal space implicitly defines a large, discrete space of goals. Its design uses some domain knowledge, but, crucially, no direct knowledge of how to solve the given task. The most common form is to use designer knowledge about which aspects of an observation are most salient. For example, many games have on-screen counters that track task-relevant quantities (health, resources, etc.); these parts of the observation are salient and can be used to guide the agent's exploration. Other examples include the inventory in MINECRAFT, sound effects in console video games, text feedback in domains like NETHACK (see Section 8.5.3 and Figure 8.2), or object attributes in robotics. In all of these cases, it is straightforward to build a set of binary functions that can be used to

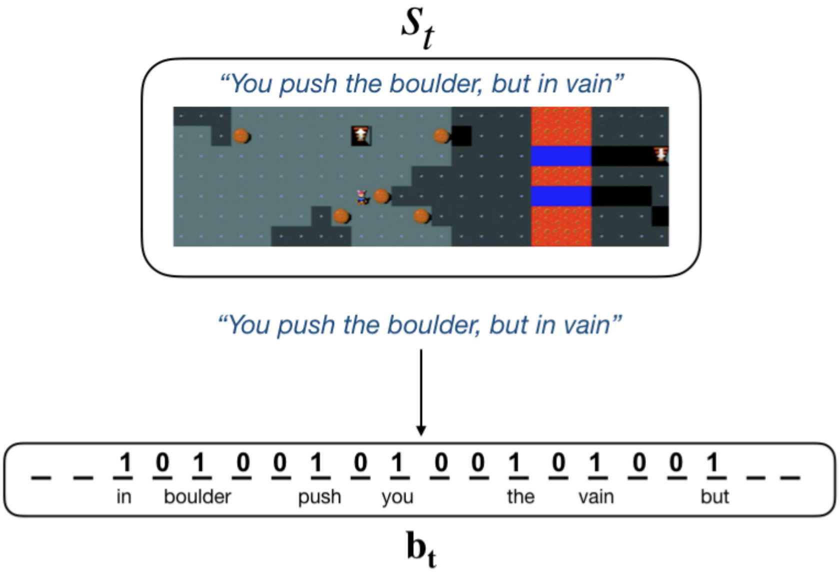


Figure 8.2: Proto-goal space in MiniHack: the message part of the observation is converted to a binary vector b_t using bag-of-words.

define the proto-goal space—for example, in NetHack, a simple proto-goal space includes one binary function for each possible word that could be present in the text feedback.

8.2.2 Representation

Each observation from the environment is accompanied by a binary proto-goal vector $\mathbf{b}_t \in \{0,1\}^{|B|}$, with entries of 1 indicating which proto-goals are achieved in the current state (Figure 8.1). Initially, the agent decomposes $\mathbf{b}_t$ into 1-hot vectors, focusing on goals that depend on a single dimension. As the agent begins to master 1-hot goals, it combines them using the procedure described in Section 8.2.3, to expand the goal space and construct *multi-hot goals*, which represent cumulative achievement of several goals.

When querying the goal-conditioned policy $\pi(a|s,g)$, we use the same 1-hot or multi-hot binary vector representation for the goal g.

8.2.3 Goal Recombinations

A neat side-effect of a binary proto-goal space B is that it can straightforwardly be extended to a combinatorially larger goal space with logical operations. For example, using the logical `AND` operation, we can create goals that are only attained once multiple separate bits of $\mathbf{b}$ are activated simultaneously.[1] One advantage of this is that it places less burden on the design of the proto-goal space, because B only needs to contain useful goal components, not the useful goals themselves. This is also a form of continual learning (Ring, 1994), with more complex or harder-to-reach goals continually being constructed out of existing ones. The guiding principle to keep this constructivist process from drowning in too many combinations is to operate in a gradual fashion: we only combine goals that, in addition to being plausible and desirable (Section 8.3), have also been *mastered* (Section 8.4.3).

8.3 Proto-goal Evaluator

The Proto-goal Evaluator (PGE) converts the large set of proto-goals to a smaller, more interesting set of goals $\mathcal{G}$. It does this in two stages: a binary filtering stage that *prunes* goals by plausibility, and a weighting stage that creates a *distribution* over the remaining goals $\mathbb{P}_{\mathcal{G}} : \mathcal{G} \to [0, 1]$, based on desirability.

8.3.1 Plausibility Pruning

Implausible proto-goals are those that most likely cannot be achieved (either *ever* or given the current data distribution). Having them in the goal space is unlikely to increase the agent's competence; to the contrary, they can distract and hog capacity. We use the following three criteria to eliminate implausible goals:

Observed: we prune any proto-goal b_i that has never been observed in the agent's experience, so far.

Reachable: we prune proto-goals that are deemed unreachable (e.g., pigs cannot fly, a person cannot be in London and Paris at the same time).

---

[1] Note that we combine goals, but not their corresponding value-functions (Barreto et al., 2019a; Tasse et al., 2021); we let the UVFA $Q_\theta(s, a, g)$ handle generalization to newly created goals.

Controllable: similarly, we prune goals that are outside of the agent's control (e.g., sunny weather is reachable, but not controllable).

For the first criterion, we simply track global counts $N(g)$ for how often we have observed the proto-goal b_i that corresponds to g being reached. Estimating reachability and controllability is a bit more involved. We do this by computing a pair of *proxy* value functions: each goal g is associated with two types of reward functions (or cumulants)—one with "seek" semantics and the other with "avoid" semantics:

$$R_{\text{seek}}(s, g) = 1 \text{ if } g \text{ is achieved in } s \text{ else } 0$$
$$R_{\text{avoid}}(s, g) = -1 \text{ if } g \text{ is achieved in } s \text{ else } 0.$$

These seek/avoid cumulants in turn induce seek/avoid policies, and value functions $V_{\text{seek}}, V_{\text{avoid}}$ that correspond to these policies. Estimates of these values are learned from transitions stored in the replay buffer $\mathcal{B}$.

A proto-goal g is **globally reachable** if it can be achieved from *some* state:

$$\max_{s \sim \mathcal{B}} V_{\text{seek}}(s, g) > \tau_1, \tag{8.1}$$

where $\tau_1 > 0$ is a threshold representing the (discounted) probability below which a goal is deemed to be unreachable.

A proto-goal g is judged as **uncontrollable** if a policy seeking it is equally likely to encounter it as a policy avoiding it:

$$\mathbb{E}_s\Big[V_{\text{seek}}(s, g)\Big] - \mathbb{E}_s\Big[-V_{\text{avoid}}(s, g)\Big] < \tau_2, \tag{8.2}$$

up to threshold τ_2. The set of plausible goals $\mathcal{G}$ is the subset of those proto-goals induced by B that satisfy both Eq. 8.1 and 8.2.

8.3.1.1 Scalably Estimating Many Seek/Avoid Values with LSPI

As a first line of defense in the process of trimming a vast proto-goal space, the reachability and controllability estimation (and hence the computation of the proxy values $V_{\text{seek}}, V_{\text{avoid}}$) must be very cheap per goal considered. On the other hand, their accuracy requirement is low: they are not used for control, and it suffices to eliminate *some* fraction of implausible goals. Consequently, we have adopted four radical simplifications that reduce the compute requirements of estimating proxy values, to far less than is used in the main deep RL agent training. First, we reduce the value estimation to a *linear* function approximation problem, by invoking two iterations of least-squares policy iteration (LSPI, (Lagoudakis and Parr, 2003; Ghavamzadeh et al., 2010)), one for the "seek" and one for the "avoid" policy. As input features for LSPI we use random projections of the observations into $\mathbb{R}^{|\phi|}$, which has the added benefit of making this approach scalable independently of the observation size. Third, the estimation is done on a batch of transitions that are only a small subset of the data available in the agent's replay buffer $\mathcal{B}$ (Lin, 1993).[2] Finally, we accept some latency by recomputing proxy values asynchronously, and only a few times (≈ 10) per minute. Section 8.5.2 shows that such a light-weight approach is indeed effective at identifying controllable goals.

8.3.2 Desirability Weighting

The second task of the PGE is to enable sampling the most *desirable* goals from the reduced set of plausible goals $\mathcal{G}$ produced via pruning. A lot has been discussed in the literature about what makes goals desirable (Gregor et al., 2016; Bacon et al., 2017; Konidaris and Barto, 2009b; Eysenbach et al., 2019a; Bellemare et al., 2016; Machado et al., 2017); for simplicity, we stick to the two most commonly used metrics: novelty and

---

[2] If the batch does not contain any transition that achieves a proto-goal, we are optimistic under uncertainty and classify it as plausible.

reward-relevance. We use a simple count-based novelty metric (Auer, 2002):

$$\text{novel}(g) := \frac{1}{\sqrt{N(g)}}, \tag{8.3}$$

where $N(g)$ is the number of times goal g has been achieved across the agent's lifetime. The desirability score (or "utility") of a goal g is then simply $u(g) := R(g) + \text{novel}(g)$, where $R(g)$ is the average extrinsic reward achieved on transitions where g was achieved. Desirability scores for each goal are turned into a proportional sampling probability:

$$\mathbb{P}_{\mathcal{G}}(g) := \frac{u(g)}{\sum_{g' \in \mathcal{G}} u(g')}. \tag{8.4}$$

In practice, when queried, the PGE does not output the full distribution, but a (small) discrete set of K plausible and desirable goals, by sampling from $\mathbb{P}_{\mathcal{G}}$ with replacement ($K = 100$ in all our experiments).

8.4 Integrated RL Agent

This section details how to integrate proto-goal spaces and PGE components into a goal-conditioned RL agent. As is typical in the goal-conditioned RL literature, we use a Universal Value Function Approximator (UVFA) neural network to parameterize the goal-conditioned action-value function $Q_\theta(s, a, g)$, which is eventually used to pick primitive actions. At the high level, we note that the PGE is used at three separate *entry points*, namely in determining how to act, what to learn about (in hindsight), and which goals to recombine. What is shared across all three use-cases is the plausibility filtering of the goal space (implausible goals are never useful). However, the three use-cases have subtly different needs, and hence differ in the goal sampling probabilities.

8.4.1 Which Goal to Pursue in the Current State

For effective exploration, an agent should pursue goals that maximize its expected learning progress, i.e., it should pick a goal that will increase its competence the most (Lopes et al., 2012). As proxies for learning progress, we adopt two commonly used heuristics, namely novelty (Auer, 2002) (Eq. 8.3) and reachability (Konidaris and Barto, 2009b). The issue with exclusively pursuing novelty is that this could lead the agent to prioritise the most difficult goals, which it cannot reach with its current policy yet, and hence induce behaviour that is unlikely to increase its competence. Thus, we combine novelty with a **local reachability** metric, for which we can reuse the goal-conditioned value $V_\theta(s_t, g)$, which can be interpreted as the (discounted) probability that the agent can achieve goal g from its current state s_t, under the current policy π_θ. To avoid computing reachability for each goal in $\mathcal{G}_t$ (which can be computationally expensive), we instead sample M goals based on novelty and pick the closest:

$$g_t = \underset{g \in \{g_1, \ldots, g_M\} \sim \text{novel}}{\arg\max} \Big[V_\theta(s_t, g) \Big].$$

8.4.1.1 Stratified Sampling over Heterogeneous Timescales

The attainment count for a goal $N(g)$ can be low because it is rarely reachable, *or* because it naturally takes a long time to reach. To account for this heterogeneity in goal space, we first estimate each goal's natural timescale and then use *stratified sampling* to preserve diversity and encourage apples-to-apples desirability comparisons. To estimate the characteristic timescale (or horizon) h for each goal, we average the "seek" value-function over the state-space: $h(g) := \mathbb{E}_{s \sim \mathcal{B}} \Big[V_{\text{seek}}(s, g) \Big]$. Once each goal has a timescale estimate, we divide the goals in the goal space into different buckets (quintiles). Then, we uniformly sample a bucket of goals; since the goals in the bucket have similar timescales ($\approx h$), we use novelty and reachability to sample a specific goal from that bucket to pursue.

8.4.1.2 Learning about Extrinsic Reward

The evaluator always picks actions to maximize the extrinsic task reward. If the actors never did during training, then the action-value function would have unreliable estimates of the task return (called the *tandem effect* (Ostrovski et al., 2021)). So, periodically, the actors pick the task reward function and select actions based on that. Since the task reward function may not correspond to a specific goal, we represent the task reward function as a special conditioning—a **0** tensor serves as the goal input to $Q_\theta(s, a, g = \mathbf{0})$.

8.4.2 Which Goals to Learn about in Hindsight

Once the agent picks a goal g to pursue, it samples a trajectory τ by rolling out the goal-conditioned policy $\pi_\theta(:, g)$. Given all the goals achieved in τ, $\mathcal{G}_A^\tau$, the agent needs to decide which goals $G \subset \mathcal{G}_A^\tau$ to learn about in hindsight.

The agent always learn about the on-policy goal g, and the task reward (which corresponds to the conditioning $g = \mathbf{0}$). Among the set of achieved goals $G \subset \mathcal{G}_A^\tau$, the agent samples a fixed set of M_{her} goals and learns about them using hindsight experience replay (Andrychowicz et al., 2017) (we use $M_{her} = 15$). Similar to the previous section, we want to sample those M_{her} goals that maximize expected learning progress. We found that using a count-based novelty score as a proxy for learning progress (sample proportionally to novel(g)) worked well for this purpose, and outperformed the strategies of (a) learning about all the achieved goals and (b) picking the M_{her} goals uniformly at random from $\mathcal{G}_A^\tau$.

8.4.3 Mastery-based Goal Recombination

We use one simple form of goal recombination in the agent: for any pair of goals that it has *mastered*, it adds their combination (logical `AND`) as a proto-goal to be evaluated by the PGE. A goal is considered mastered when its success rate is above a pre-specified threshold κ ($= 0.6$ in all our experiments). For example, if the agent has mastered the goal of getting the key, and another goal of reaching the door, it will combine them to

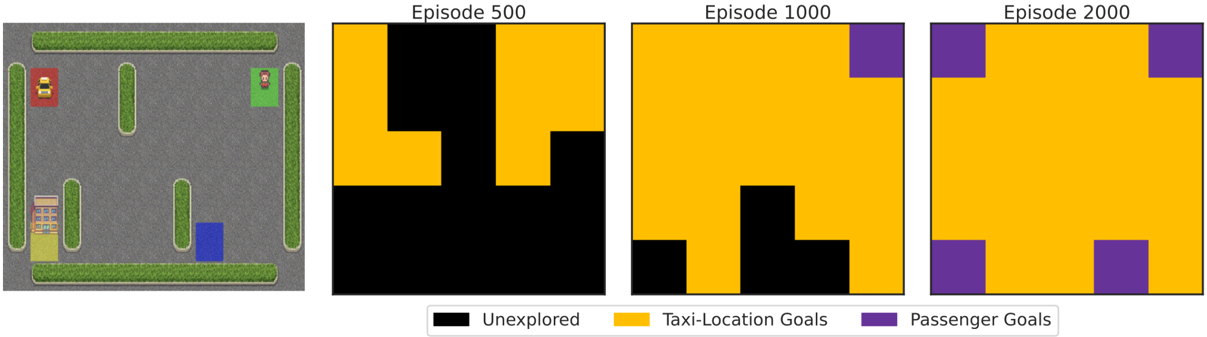


Figure 8.3: Progression of the goal space refinement in SPARSETAXI (domain illustrated in the leftmost sub-figure (Brockman et al., 2016)). This is a visualization of the 5×5 grid; yellow and black squares are explored and unexplored taxi locations respectively; purple squares denote explored passenger locations. The set of plausible goals grows over time from controlling the taxi location, to eventually controlling the location of the passenger. The passenger destination is always deemed uncontrollable by the Proto-goal Evaluator.

create a new proto-goal which is attained when the agent has reached the door with the key.

8.4.4 Distributed RL Agent Implementation

For all non-tabular experiments, we integrate our method with an off-the-shelf distributed RL agent, namely R2D2 (Kapturowski et al., 2019). It is a distributed system of 128 actors asynchronously interacting with 128 environments. The learner is Q-learning-based, using a goal-conditioned action-value function $Q_\theta(s, a, g)$ parameterized as a UVFA. Experience is stored in a replay buffer $\mathcal{B}$, including the binary proto-goal annotation vector $\mathbf{b}$.

8.5 Experiments

Our empirical results are designed to establish proto-goal RL as an effective way to do exploration, first in a classic tabular set-up (TAXI, Section 8.5.1), and then in two larger-scale domains (NETHACK and BABA IS YOU, Sections 8.5.3 and 8.5.4) whose combinatorial proto-goal spaces, left unpruned, would be too large for vanilla goal-conditioned RL. Ablations and probe experiments show the effectiveness of our controllability and

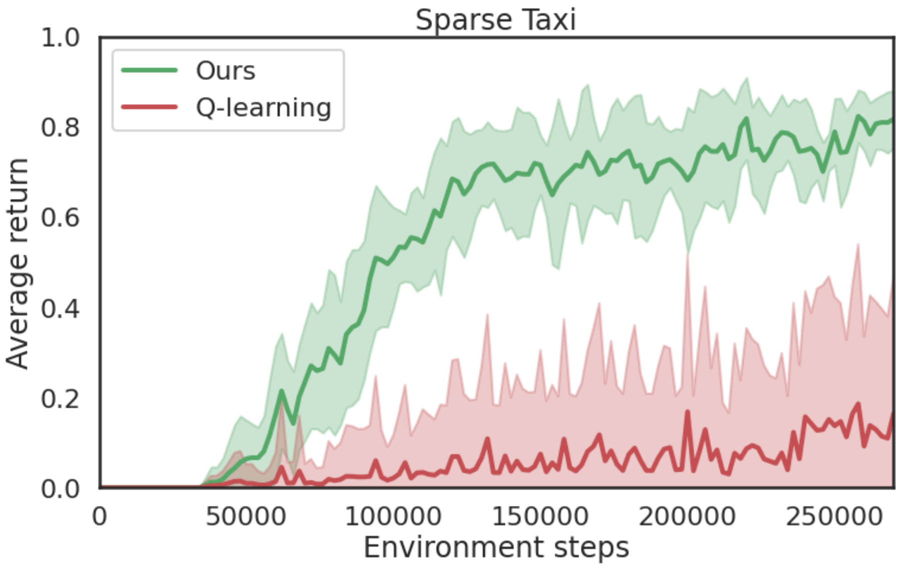


Figure 8.4: Learning curves comparing Q-learning with ϵ-greedy exploration to proto-goal-based exploration in SPARSETAXI. Error bands denote standard error over 20 independent runs.

desirability metrics, and provide qualitative insights into the discovered goal spaces.

8.5.1 Tabular Experiment: Exploration in Taxi

We build intuition about proto-goal exploration on the TAXI domain, a tabular MDP classically used to evaluate hierarchical RL algorithms (Dietterich, 2000). In this problem, the agent controls a taxi in a 5×5 grid; the taxi must first navigate to the passenger, pick them up, take them to their destination (one of 4) and then drop them off. The default reward function is *shaped* (Randløv and Alstrøm, 1998), but to make it a harder exploration problem, we propose the SPARSETAXI variant, with two changes : (a) no shaping rewards for picking up or dropping off the passenger and (b) the episode terminates when the passenger is dropped off. In other words, the only (sparse) positive reward occurs when the passenger is dropped off at their correct destination.

As the proto-goal space, we use a factoring of the state space, namely one b_i for each entity (taxi, passenger, destination) in each grid location ($|B| = 34$). Figure 8.3 shows the progression of how the PGE gradually refines a goal space throughout training. The set

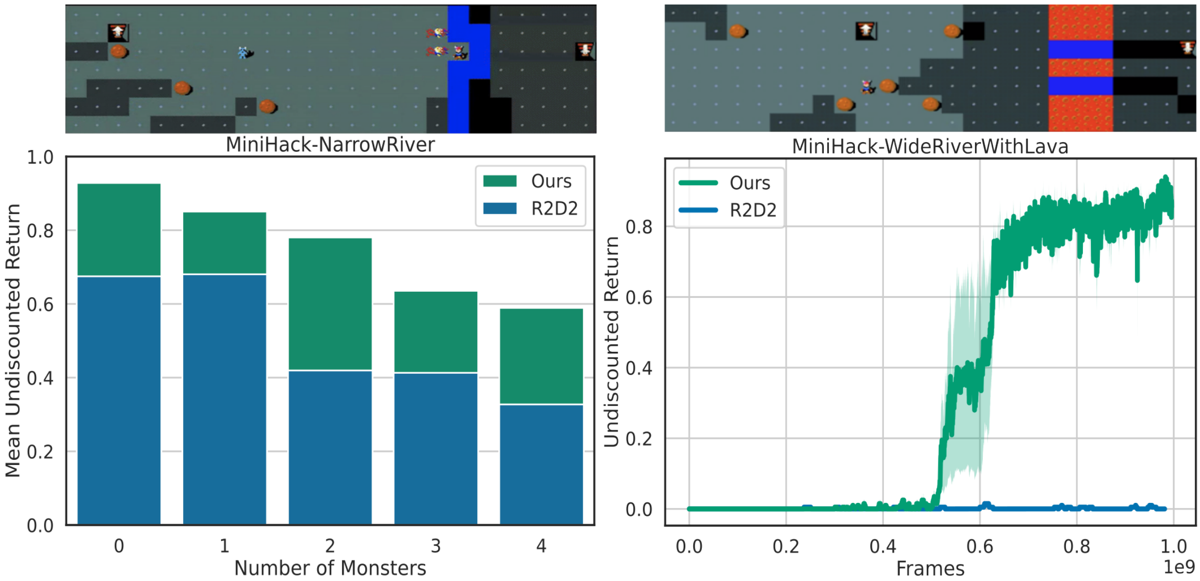


Figure 8.5: MINIHACK experiments. **Left**: Final performance, when varying difficulty via the number of monsters in NARROWRIVER. **Right**: The challenge of getting off the ground in WIDERIVERWITHLAVA (no monsters). In all $5 + 1$ scenarios, using proto-goals outperforms the baseline R2D2 agent. Scenarios are illustrated above the result plots.

of reachable states expands gradually, mimicking a curriculum; at first, goals correspond to navigating the taxi to different locations, later they include goals for dropping off the passenger at different depots. Also noteworthy is that proto-goals corresponding to the destination are absent from the goal space, because they are classified as uncontrollable.

In terms of performance, our proposed method of goal-discovery also leads to more sample-efficient exploration in SPARSETAXI (Figure 8.4). Compared to a vanilla Q-learning agent with ϵ-greedy exploration, our goal-conditioned Q-learning agent learns to reliably and quickly solve the task.

8.5.2 Verifying the Controllability Measure

Our method of measuring controllability using the discrepancy between “seek” and “avoid” values (Section 8.3.1) is novel, hence we conduct a set of sanity-checks to verify that it can capture controllability in many of its guises. Three experiments probe three separate types of controllability:

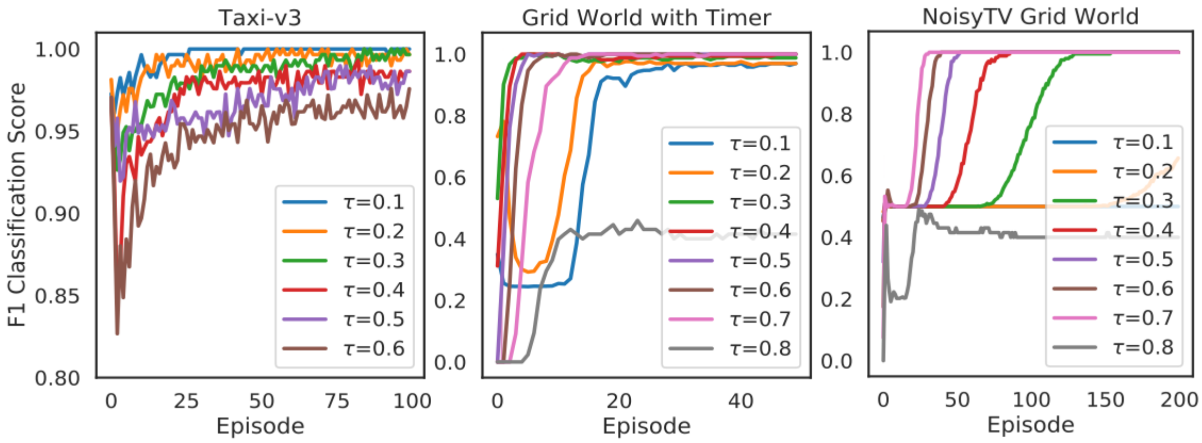


Figure 8.6: Testing our seek-avoid controllability measure using the F1 classification metric, which is the harmonic mean of precision and recall (higher is better). In each domain, we collect transitions using a random policy. For evaluation, we manually specify which of the proto-goals are controllable vs uncontrollable. Then, we measure how effectively the agent can classify controllable vs uncontrollable proto-goals for different values of the threshold τ_1 (see Equation 8.2).

Static controllability: Proto-goals whose attainment status does not change. The passenger destination in TAXI is a good example of this kind of uncontrollability—while the destination changes *between episodes*, there is no *single transition* in the MDP in which it changes.

Time-based controllability: Some problems have a *timer* that increments, but is not controllable by the agent. We check whether our controllability metric classifies such time-based proto-goals as plausible, using 4×4 gridworld with a timer that increments from 1–100 (which is the max number of steps in an episode).

Distractor controllability: More generally, parts of the observation that change independently of the agent's actions are distractors for the purpose of controllability. For this test, we use a visual gridworld, where one image channel corresponds to the controllable player, and the two other channels have pixels that light up uniformly at random (Gregor et al., 2016) (often referred to as a "noisy TV" (Schmidhuber, 2010a; Burda et al., 2019a)).

For each of these setups, we compare our controllability predictions (Eq. 8.2) to ground-

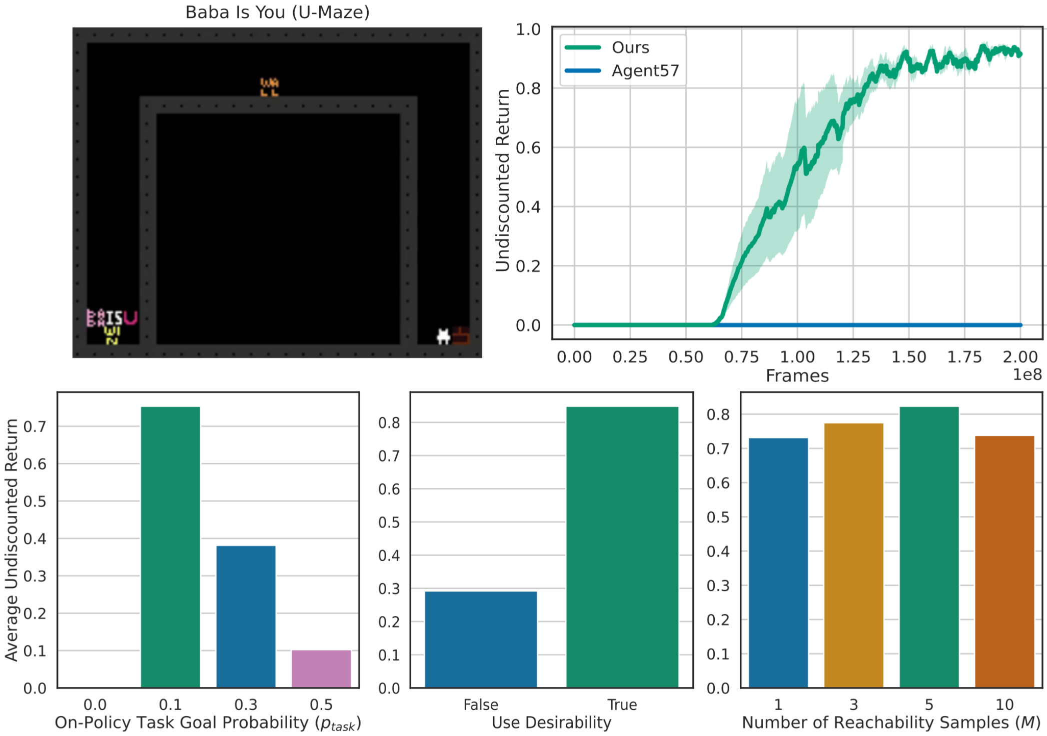


Figure 8.7: Baba Is You experiments. **Top left**: U-Maze domain: the agent controls the white sheep at the bottom-right, which must move the "wall" block just above the "is" block in the bottom-left and then go to the brown colored wall in the bottom-right. **Top right**: Learning curves comparing our approach to (flat-lining) Agent57. **Bottom**: Ablations showing the importance of sometimes acting according to the task reward during training *(left)*, using desirability metrics in the PGE *(middle)*, and the impact of the number of goals sampled for computing local reachability *(right)*.

truth labels, and find it to correctly classify which proto-goals are controllable (Figure 8.6). As the agent interacts with the environment for more episodes, its prediction quality improves, suggesting dependence on the amount of data used to estimate the seek/avoid values. Furthermore, these results suggest that for most values of τ_1, the agent can correctly identify controllable proto-goals, but the value of τ_1 depends on the environment.

8.5.3 Natural Language Proto-goals: MiniHack

The first non-tabular domain we investigate is MiniHack (Samvelyan et al., 2021), a set of puzzles based on the game NetHack (Küttler et al., 2020), which is a grand challenge in RL. In addition to image-based observations, the game also provides natural language messages. This space of language prompts serves as our proto-goal space—while this space is very large (many 1000s of possible sentences), it contains a few useful and interesting goals that denote salient events for the task. Figure 8.2 illustrates how word-based proto-goal vectors are created in MiniHack.

I used two variants of the River task as exemplary sparse-reward exploration challenges: in this task the agent controls a character that must cross a river to get to the task-goal; to cross the river, the character must first construct a bridge out of boulders. I chose these tasks because the survey of Samvelyan et al. (2021) noted that while novelty-seeking algorithms (Burda et al., 2019b; Raileanu and Rocktäschel, 2020) could solve the easiest version of River, they were unable to solve more difficult variations.

In all of these, the agent must make a bridge out of boulders and then cross it to reach its goal. In the NarrowRiver variant, the agent needs to place one boulder to create a bridge, and the difficulty depends on the number of monsters who try to kill the player. Figure 8.5 *(left)* shows that while increasing the number of monsters degrades performance, our proto-goal agent outperforms the baseline R2D2 agent on each task setting. In the WideLavaRiver variant, the river is wider, requiring 2 boulders for a bridge, and includes deadly lava that also dissolves boulders. Figure 8.5 *(right)* shows that our proto-goal agent comfortably outperforms its baseline.

Discovered goal space. Words corresponding to important events in the game find their way into the goal-space. For instance, the word "water" appears in the message prompt when the boulder is pushed into the water and when the the player falls into the river and sinks. Later, combination goals like "boulder" `AND` "water" also appear in the goal-space and require the agent to drop the boulder into the water.

8.5.4 Doubly Combinatorial Puzzles: Baba Is You

The game Baba Is You (Teikari, 2019) has fascinating emergent complexity. At first sight, the player avatar is a sheep ("Baba") that can manipulate objects in a 2D space. However, some objects are "word-blocks" that can be arranged into sentences, at which point those sentences become new rules that affect the dynamics of the game (e.g., change the win condition, make walls movable, or let the player control objects other than the sheep with an "X-is-you"-style rule). The natural reward is to reach the (sparse) win condition of the puzzle after 100s of deliberate interactions.

When testing various RL agents on Baba is You, we observed a common failure mode: the exploration process does not place enough emphasis on manipulating object and text blocks. So, we created a simple level (U-Maze shown in Figure 8.7) that is designed to focus on the crucial aspect of rule manipulation. This puzzle requires the agent to learn how to push a word block in place (from center to bottom left), which adds a new win-condition, and then touch the correct block (on the bottom right). Exploration here is challenging because the agent has to master both navigation and block-manipulation before it can get any reward. In addition, the game's combinatorially large state space is a natural challenge to any novelty-based exploration scheme.

As in Taxi, we use a simple factored proto-goal space, with one binary element for every object (specific word blocks, wall, sheep) being present at any grid-position. Plausible 1-hot goals could target reaching a specific position of the sheep or movable blocks. Most combinations (2-hot proto-goals) are implausible, such as asking the sheep to be in two locations at once, but some could be useful, e.g., targeting particular positions for both "Baba" *and* a word-block.

Given the exploration challenges in this domain (R2D2 never sees any reward, even on smaller variants of the puzzle), we use the stronger, state-of-the-art Agent57 agent as baseline here, which adds deep exploration on top of R2D2—it constructs an intrinsic reward using novelty and episodic memory (Badia et al., 2020a). Figure 8.7 *(top right)* shows that our R2D2 with proto-goals (but no intrinsic rewards) outperforms Agent57. Note that with careful tuning, Agent57 does eventually get off the ground on this task, but never within the 200M frame budget considered here. On the other hand, Agent57 has the advantage that it does not require engineering a proto-goal space.

Discovered goal space. At first, the goal-space is dominated by navigation goals; once these are mastered, goals that move the word-blocks begin to dominate. Then the agent masters moving to a particular location *and* moving a word-block to some other location. Eventually, this kind of exploration leads to the agent solving the problem and experiencing the sparse task reward.

Ablations. Figure 8.7 *(bottom left)* analyzes how often the agent should act according to the extrinsic reward instead of picking a goal from the discovered goal-space. When that probability is 0, the agent never reaches the goal during evaluation; acting according to the task reward function 10% of the time during training performed the best in this setting. In a second ablation, Figure 8.7 *(bottom middle)* shows the importance of using desirability metrics on top of plausibility when mapping the proto-goal space to the goal-space. Finally, Figure 8.7 *(bottom right)* shows the impact of the number of goals sampled for computing local reachability during goal-selection (Section 8.4.1).

8.6 Conclusion and Future Work

We presented a novel approach to using goal-conditioned RL for tackling hard exploration problems. The central contribution is a method that efficiently reduces vast but meaningful proto-goal spaces to a smaller sets of useful goals, using plausibility and desirability criteria based on controllability, reachability, novelty and reward-relevance. Directions for future work include generalising our method to model-based RL to plan with jumpy goal-based models, more fine-grained control on when to switch goals (Pislar et al., 2022), making the proto-goal space itself learnable, as well as meta-learning the ideal trade-offs between the various desirability criteria.

CHAPTER 9

Going Beyond State-Reaching: Learning Abstractions for Intrinsically Motivated Skill Discovery

Recall successfully riding your bicycle for the first time. The moment you balanced your bicycle without training wheels, you realized that you had done something right. In future practice, you recreated your speed and balance, not the color of your t-shirt, the weather conditions outside or the exact wear and tear on your tires. However, when existing skill discovery algorithms attempt to learn a policy to reach a previously encountered event (Chentanez et al., 2005), they attempt to recreate *every aspect* of that state. Examples include the ϵ-ball from deep skill graphs (Bagaria et al., 2021b), pixel-wise equality (Veeriah et al., 2018), the perceptual hashing from Go-Explore (Ecoffet et al., 2021), random projections of the target state (Dabney et al., 2021; Farebrother et al., 2023), or a neural classifier that predicts perceptual similarity between states (Vezhnevets et al., 2017; Hafner et al., 2022). This state-reaching strategy adversely impacts hierarchical agents in several ways. First, the resulting skills are nontransferable; consider a robot grabbing a cup from a cluttered table: the skill's subgoal is tied to the position of every object in the state, requiring the agent to learn combinatorially more skills. Beyond stymying transfer, which is one of the key benefits of HRL (Taylor and Stone, 2009), state-reaching artificially reduces the size of the skill's subgoal region, which complicates policy learning and eventually makes the skill less useful to the agent. Finally, recreating a state is *unscalable* because in vast, realistic domains, recreating all aspects of a particular state may be impossible (Bagaria and Schaul, 2023).

Our skill discovery algorithm is inspired by intrinsic motivation in humans and other animals: during play, when children cause something interesting to happen, they try recreate it, with increasing efficiency (White, 1959). They practice this skill until they get bored and move on, but retain the ability to reuse the acquired skill later (Barto et al., 2004). Similarly, when during an AI agent's exploration, it notices something particularly interesting, it should learn a skill to reliably achieve it in the future. Other algorithms (for example, Bagaria et al. 2021b) use these ideas; but resort to a state-reaching objective. Instead, we identify the subset of the state features responsible for the saliency (Chentanez et al., 2005) of that state, and create a subgoal achievement classifier that attends to those specific features.[1] These classifiers serve as pseudo-reward functions for training skill policies, which are composed together by a high-level policy over options to maximize extrinsic reward.

We test our algorithm in two challenging exploration problems: MiniGrid-KeyCorridor (Chevalier-Boisvert et al., 2023), and VisualTaxi (Dietterich, 2000; Allen et al., 2021); both have image-based observations, and a sparse reward function. We quantitatively compare our agent with a novelty maximization technique (Lobel et al., 2023) and a flat model-free RL algorithm (Kapturowski et al., 2019) based on Q-learning (Watkins and Dayan, 1992) with ϵ-greedy exploration and find that our algorithm performs favorably.

---

[1] We do *not* assume that state features are given and learn subgoal achievement classifiers directly from images, but the language of "features" helps present our ideas simply.

9.1 Background and Related Work

Our ultimate goal is still to build agents that maximize rewards in Markov Decision Processes (MDPs). To aid in reward maximization in large environments with sparse rewards, our agent will discover options, where each option is described using its initiation function, policy and termination function: $o \in \mathcal{O} = (I_o, \pi_o, \beta_o)$. Each option is concerned with its own subgoal, and the agent's policy over options $\pi_{\mathcal{O}}$ decides which option to execute with the aim of maximizing the task reward.

Intrinsically Motivated Skill Discovery. In this chapter, we use intrinsic motivation (IM) as a basis for option discovery. IM is a drive to do something for its own sake; skills learned using IM may not have immediate, obvious utility, which often only becomes clear later on. Within computational RL, IM has been estimated using several proxies: novelty, surprise, incongruity, empowerment, learning progress, compression, and so on; we quantify IM using novelty, but our approach is compatible with other proxies of IM as well. Crucially, IM provides a means for learning a collection of reusable skills to be used as building blocks for systematic exploration of large environments.

9.1.1 Learning Subgoal Achievement Classifiers

During option discovery, the subgoal g_o of each option $o \in \mathcal{O}$ must be defined. This subgoal is a function that maps a state s to a binary decision that indicates whether or not the subgoal g_o is satisfied in s. The subgoal is turned into a pseudo-reward R_o and termination function β_o that is used to train that option's policy π_o: execution terminates with a reward of 1 when the subgoal is achieved and continues without reward otherwise. Most option discovery algorithms fall in one of two categories: they either require a system designer to manually define a subgoal achievement function, or they resort to state-reaching, where the benefits of abstraction start to vanish. Few methods take on the challenge of learning a subgoal achievement function during the discovery process; three approaches are most common:

- **Perceptual similarity.** Commonly, algorithms compare the distance between the current state s_t and the goal state g_t. This is either done using euclidean distances in the raw observation space (with a fixed, pre-specified threshold), or by first computing random projections of s_t and g_t and comparing the distance in the projected space. Hashing the state and goal state using domain-specific perceptual metrics is also a common strategy. Autoencoders are also used to compute distances between lower dimensional embeddings of the state and the goal state, but these autoencoders are trained using pixel-wise reconstruction errors, which results in classifiers that are tied to all perceptual features of the inputs.
- **Temporal distance estimation.** These methods learn a classifier that predicts whether the current state is less than K time steps away from a goal state (K is a fixed hyperparameter). Such a classifier can be trained on states encountered during learning: for every trajectory $\tau = (s_1, s_2, s_3, ...)$, any two pairs of states that are less than K steps apart are positive examples and all other pairs constitute negative examples; the classifier $g_\theta : S \times S \rightarrow \{0, 1\}$ is trained using cross-entropy loss. While thresholding using the number of time steps is more natural than thresholding pixel-wise differences, such a classifier still attends to all aspects of the goal state.
- **Mutual information estimation.** DISCERN (Warde-Farley et al., 2019a) computes the mutual information between the next state s_T of the option and the goal state g_t as the pseudo-reward function for training that option's policy. This has the advantage that it abstracts away uncontrollable aspects of the state, but it has two disadvantages: (a) the maximum mutual information between s_T and g_t is unknown, so it is unclear *when* to terminate the option, and (b) features of the state that are controllable might still be irrelevant for that particular option's subgoal, but such features are not abstracted away by this technique.

These techniques differ in their level of sophistication and ease of implementation, but they all fall in the category of *state-reaching*: no explicit effort is made to identify a few salient features of state the and subsequently learn options that only try to achieve certain values in those features.

9.1.2 Options of feature attainment

The strategy of learning options for different features has been explored by some HRL methods. For example, in STOMP, each option is responsible for separately maximizing a particular state feature (Sutton et al., 2024). Additionally, Proto-goal RL (Chapter 8), combines existing features to learn options that achieve combinations of features (Bagaria and Schaul, 2023). However, both these methods assume that a factored representation is provided to the agent, while our work discovers the factored goal-space autonomously via interaction data.

9.2 Method: Abstract Subgoal Discovery

Modern reinforcement learning systems can efficiently compute proxies for intrinsic motivation, like novelty (Kapturowski et al., 2022). However, their use as a guiding signal for option discovery focuses on state-reaching, which fails to scale beyond modest navigational problems (Hansen et al., 2021). To learn new skills, our agent first executes a low-level exploration policy that tries to visit novel states; if it observes a particularly novel state during this interaction, it compares the most novel state to the most boring state in that trajectory. Then, the agent asks counterfactual questions: if each feature in the most novel state was reset to its value in the boring state (one-by-one), how much does the novelty of the resulting counterfactual state drop? Feature resets that result in large changes to novelty are responsible for high novelty predictions. After repeating this process for each feature, we are left with the minimal set of features that explains the difference in novelty between the most novel and most boring state (illustrated in

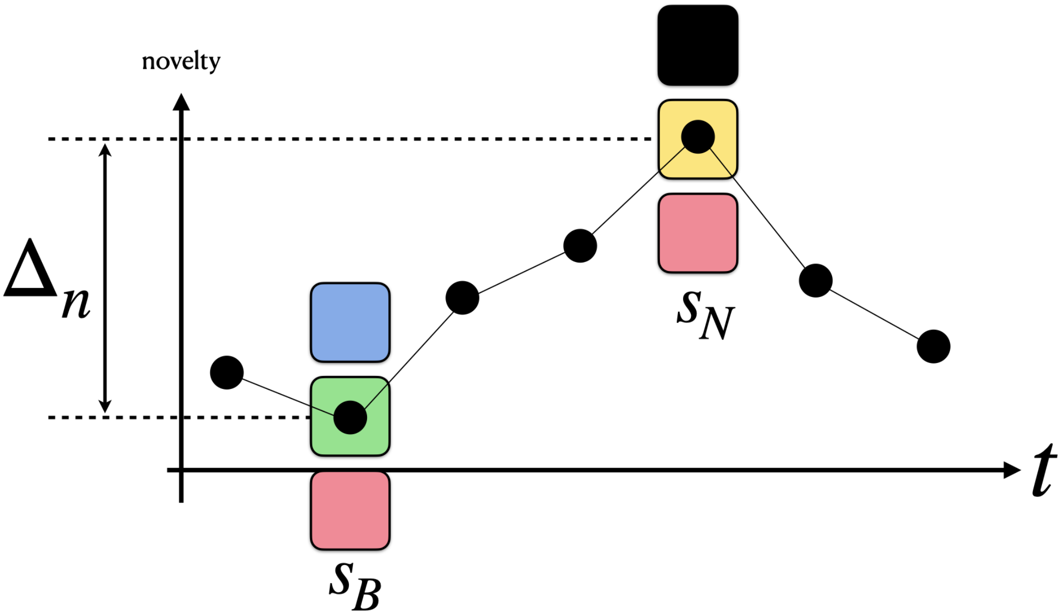


Figure 9.1: **Illustration of the core insight.** Suppose an agent observes a state trajectory of length 7 and a novelty estimator assigns each state s_t in that trajectory a novelty score $n_\phi(s_t)$. s_N denotes the most novel state, and s_B denotes the most boring state in that trajectory. Suppose that each state has 3 features; the colored boxes represent feature values for those two states. Rather than simply treating s_N as a target state, we seek to identify which features in particular were responsible for the large difference in novelty $\Delta_n = n_\phi(s_N) - n_\phi(s_B)$. Our algorithm can handle input images, but we show state features here for simplicity.

Figure 9.2). Finally, a classifier is created that only attends to the discovered feature set, ignoring the rest. This classifier serves as an *abstract* subgoal achievement classifier and the agent subsequently learns a skill policy to achieve that subgoal. This process is illustrated in Figure 9.1.

Our agent repeatedly executes the following high-level steps: (a) identify the subgoal with the highest potential for exploration, (b) achieve that subgoal by executing the corresponding option, (c) explore from that subgoal after achieving it, (d) identify the most (locally) interesting state in that trajectory, (e) determine which features make that state interesting, (f) initialize a new option to achieve the new subgoal, and add it to the agent's set of options.

9.2.1 Generating Exploratory Behaviors

For effective low-level exploration, we follow the same strategy as the IM-DSG agent from Chapter 7. We first select an existing subgoal from where we expect to see the most novel states. After achieving that subgoal using a goal-conditioned policy, we execute a policy that is trained to maximize a novelty-based reward function. While there are many candidate methods that can approximately compute novelty, we use CFN (Lobel et al., 2023) that trains a function f_ϕ that maps a state to a novelty score between 0 and 1.

Where to explore from? We first assign each existing subgoal a score that reflects the utility of exploring from it:

$$U(n) = \mathbb{E}_{s \sim n}\left[V_{\text{novelty}}(s)\right] \approx \mathbb{E}_{s \sim n}\left[V_{\text{CFN}}(s)\right], \forall n \in \mathcal{N} \tag{9.1}$$

where $\mathcal{N}$ is the set of all subgoals discovered so far, $V_{\text{CFN}}(s)$ is the CFN value function that accumulates the novelty reward function $R_{\text{CFN}} = f_\phi(s)$ and the expectation is taken over all the states in which that subgoal classifier has been triggered so far. Once each subgoal has been assigned a utility, we simply sample one from the following distribution:

$$\mathbb{P}_n = \frac{U(n)}{\sum_{n' \in \mathcal{N}} U(n')}. \tag{9.2}$$

How to conduct the exploration? Once a subgoal n^* is sampled from $\mathbb{P}_n$, a goal-conditioned policy $\pi_\theta(s_t|n^*)$ is executed to achieve it. If the goal-conditioned policy is successful, the CFN policy π_{CFN} is executed.

9.2.2 Identifying Interesting Events

The CFN policy execution results in a trajectory $\tau = \{s_1, s_2, .., s_K\}$. If there is any state in that trajectory that is particularly novel, then we begin our search for a new subgoal classifier. First, the agent identifies the most novel state s_N and most boring

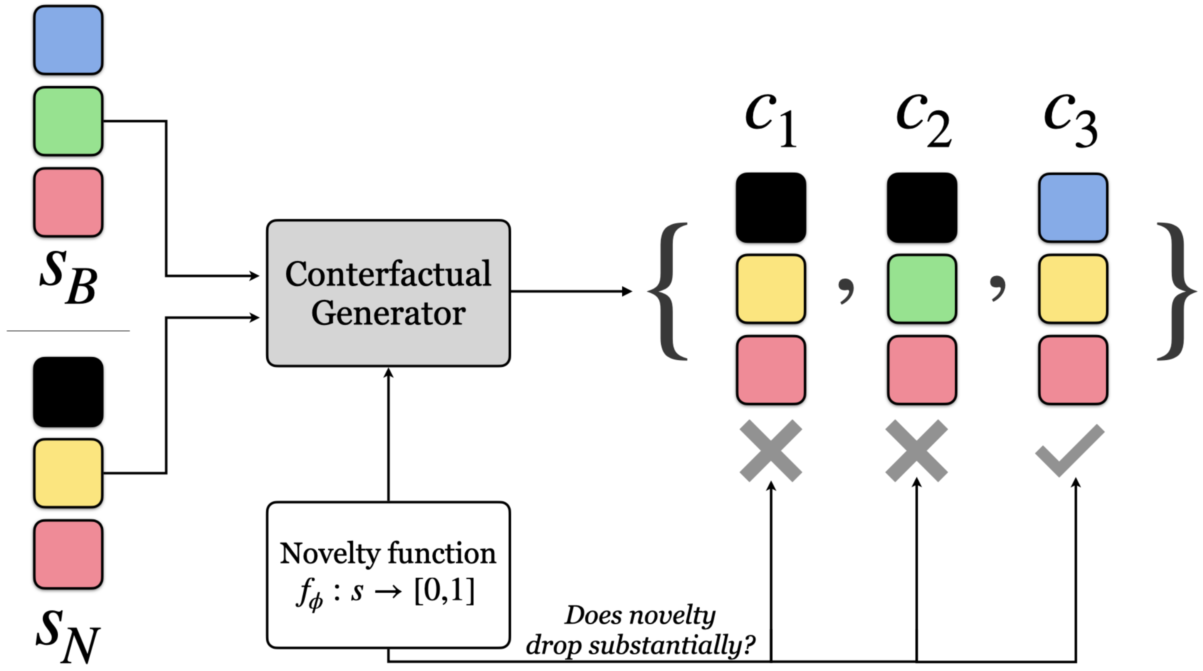


Figure 9.2: **Illustration of our feature selection algorithm.** The most boring and most novel states, s_B and s_N respectively, are input into the Counterfactual Generator, which outputs 3 counterfactual states: c_1, c_2, c_3. Changing the third feature (pink→ pink) does not impact novelty, restoring the second feature from yellow to green does not cause *enough* of a drop, but changing the first feature from black to blue substantially lowers novelty, indicating that the first feature (black) accounts for most of the novelty of s_N.

state s_B in the trajectory:

$$\text{if } \max_s \Big[f_\phi(s)\Big] > \mu_t + \sigma_t, s_N = \arg\max_{s\in\tau} \Big[f_\phi(s)\Big] \text{ and } s_B = \arg\min_{s\in\tau} \Big[f_\phi(s)\Big].$$

where μ_t and σ_t are the running mean and standard deviation of the CFN rewards encountered so far.

9.2.3 Feature Selection Algorithm

After identifying the most novel and most boring states (s_N and s_B) from the previous step, the agent identifies the minimal set of features that explains the difference in novelty between s_N and s_B. The feature selection algorithm, illustrated in Figure 9.2, is composed of two high-level steps: (a) counterfactual generation and (b) feature elimination.

9.2.3.1 Counterfactual Generation

When the states are already factored feature vectors (for e.g, in MuJoCo), then the counterfactual generation is straightforward: given s_N and s_B, replace each feature of s_N with its corresponding value in s_B, one-by-one. When the states are represented as images, the process is more involved and is described via Algorithm 2.

Algorithm 2 Counterfactual Generator for Image-based Observations

Inputs: Exploration trajectory τ, most novel state s_N and most boring s_B
Output: Bounding boxes in s_N that reflect the relevant parts of s_N.

```
 1: Initialize output list of counterfactual images C as an empty list.
 2: Compute background image $b$ using trajectory $\tau$.
 3: Remove background from $s_N$ and $s_B$ to get $\bar{s}_N, \bar{s}_B$.
 4: Find contours $\mathcal{C}_B$ in $\bar{s}_B$.
 5: Construct bounding boxes $\mathcal{B}_B$ around the contours $\mathcal{C}_B$.
 6: For each bounding box in $\mathcal{B}_B$, find the corresponding patch in $\bar{s}_N$.
 7: Create a mapping from each bounding box $b_B \in \mathcal{B}_B$ to its most similar bounding box
    patch $b_N \in \mathcal{B}_N$.
 8: for Bounding box $b_B \in \mathcal{B}_B$ do
 9:     Initialize the counterfactual image c with $s_N$.
10:     Replace the patch $b_B$ of $c$ with the corresponding patch in $s_B$: c[$b_B$] $= s_B$[$b_B$].
11:     if $b_B$ has a corresponding bounding box in $\mathcal{B}_N$ then
12:         Replace the patch $b_N$ of $c$ with the corresponding patch in $s_B$: c[$b_N$] $= s_B$[$b_N$].
13:     end if
14:     return Add counterfactual image c to output list C.
15: end for
16: return Counterfactual image list C.
```

In Algorithm 2, the Counterfactual Generator computes a background (using KNN Background Subtraction) and subtracts it from s_N and s_B. Then, it detects contours in the resulting images (Line 4) using the Suzuki-Abe algorithm (Suzuki and Abe, 1985). Bounding boxes are constructed around these contours and matched between the two images (the matching is done via a cross-correlation based template matching algorithm). These are well established, basic computer vision algorithms with efficient implementations in the popular open-source library `OpenCV` (Bradski, 2000).

Limitations. Our approach to counterfactual generation exploits the simplicity of the image observations in our test domains. We found these techniques to be easy to implement and effective in our experiments, but future work should investigate more elegant and scalable solutions; the works of Su et al. (2024) or Radford et al. (2021) could be good starting points.

9.2.3.2 Feature Elimination

After obtaining a set of counterfactual states from the previous step, we input them one-at-a-time to the novelty function f_ϕ. When the predicted novelty of a counterfactual state is significantly lower than the novelty of the entire state s_N, then we consider the corresponding feature to be salient; we retain all such features and discard the rest.

9.2.4 Classification

Having identified a minimal set of features that explain the novelty difference Δ_n between s_B and s_N, we construct a classifier $g : s \rightarrow \{0, 1\}$ that describes the subtask of a new option. This classifier is severely constrained in terms of its training data—s_N is the only state that we can confidently label as a positive example. Because of the severely constrained (positive) dataset size, we take a non-parametric approach to build our classifier.

Classifier representation. Each classifier g, is associated with two attributes. The first attribute is the positive example for the classifier—the most (locally) novel state, s_N. The second attribute is the feature extraction function obtained from the previous step of the algorithm (Section 9.2.3). In the case of already factored state representations (e.g, in MuJoCo), this feature extraction function simply extracts the relevant features of a state, i.e, $h : S \in \mathbb{R}^D \rightarrow \mathbb{R}^d$, where $d < D$. When the input observations are in the form of images, then h extracts the bounding boxes retained by the feature selection algorithm.

Making classification decisions. Given an arbitrary state s, g needs to output a binary classification label. The first step is to apply the feature extractor to g's positive example s_N. The second step is to then compare s_N's extracted features with the features extracted from the input state s:

$$g(s) = \mathbb{I}\Big\{\mathcal{D}(h(s_N), h(s)) < \xi\Big\},$$

where $\mathbb{I}$ is the indicator function, ξ a hyperparameter threshold, and $\mathcal{D} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is a function that compares the dissimilarity between two input feature vectors. When the input space is factored, $\mathcal{D}$ is the Euclidean distance metric. When the input space is images, then $\mathcal{D}$ is the cross-correlation between the extracted patches from the two input images; this subroutine is a simple call to `OpenCV`'s `matchTemplate(x, y)` function (Bradski, 2000). Again, this approach exploits the visual simplicity of most RL benchmark problems, but may not be a scalable choice for complex natural images, where neural approaches might be a better fit. Since the computer vision algorithm is not our core contribution, we chose the simplest algorithm that worked in our experimental suite.

9.2.5 Policy Learning

Once a classifier g is constructed, the agent creates a new option which gets a terminating reward of 1 for reaching a state for which g returns true, and 0 otherwise. This option is added to the agent's option space $\mathcal{O}$ and can be executed by the policy-over-options $\pi_{\mathcal{O}}$ in the future. The option's policy π_o is learned using a goal-conditioned version of R2D2 (Kapturowski et al., 2019) with hindsight experience replay (Andrychowicz et al., 2017).

9.3 Experiments

We begin by performing probing experiments to qualitatively understand how the subgoals discovered using our method differ from those learned using other popular

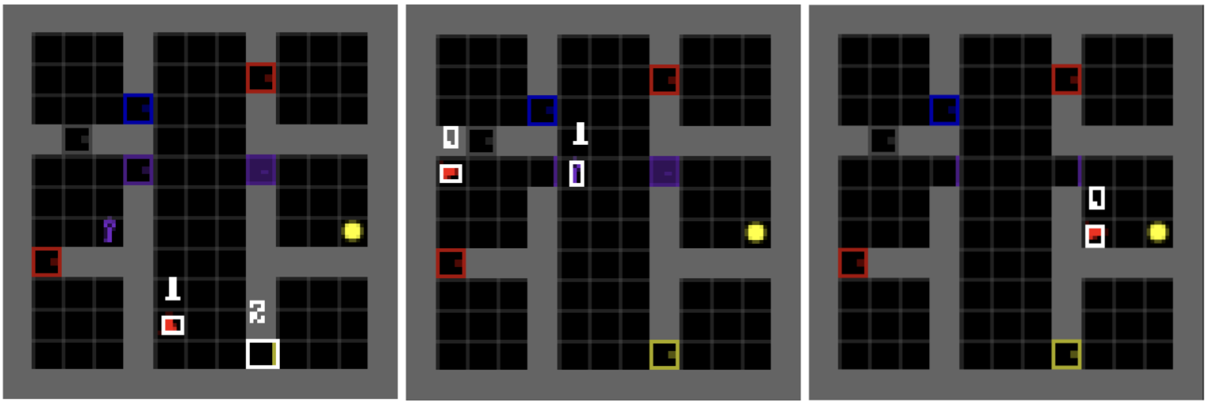


Figure 9.3: **Subgoal achievement classifiers** learned in MiniGrid-KeyCorridor. Visualized bounding boxes show the patches that were deemed relevant for each of the three subgoal classifiers shown, all other pixels are ignored. *(left)* this option's subgoal depends on reaching a particular location and opening the yellow door; *(middle)* this option aims to reach a particular location in the mid-left room but while the key is in the hallway; *(right)* this option aims to put the player inside the locked room with the ball (the task goal), regardless of the position of the key or the states of any of the doors.

methods. Our experimental suite consists of:

- **VisualTaxi.** This is a more challenging version of the classic Taxi problem (Dietterich, 2000): the agent controls a taxi in a 10 × 10 grid, which has 7 depots; a passenger awaits in one of the depots and needs to be dropped off in a different depot (Diuk et al., 2008). If the agent successfully completes the entire task, it gets a sparse terminating reward of 1. Additionally, the agent gets image observations of the domain (Allen et al., 2021), rather than a tabular or factored state as in previous work (Bagaria and Schaul, 2023).
- **MiniGrid-KeyCorridor.** Same as in Chapter 7.

9.3.1 Qualitative Analysis

Figure 9.3 shows the subgoals for 3 different options discovered in MiniGrid. While state-reaching options would attend to the configuration of all objects in this environment, our agent only attends to a small handful. This results in an agent with fewer options, which makes the learning problem of the policy on options $\pi_{\mathcal{O}}$ easier.

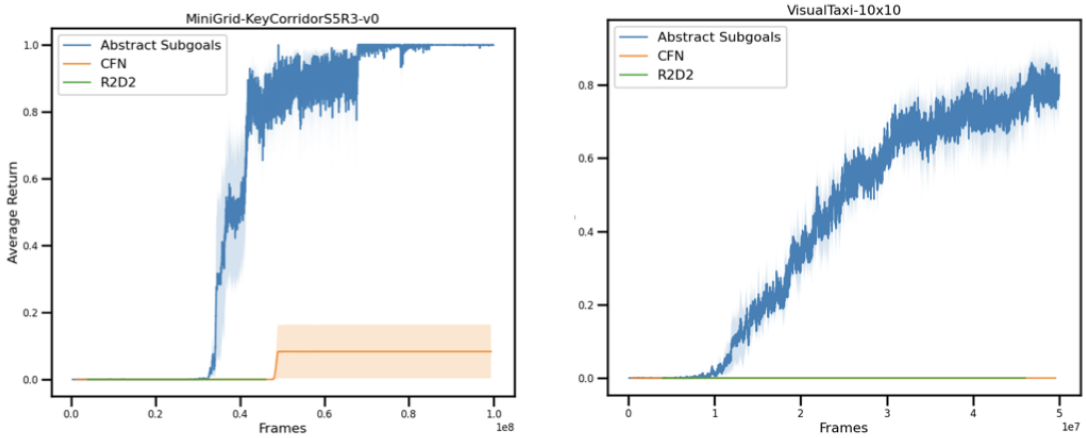


Figure 9.4: Learning curves comparing our agent (Abstract Subgoals) with a non-hierarchical novelty maximizing RL method (CFN) and a vanilla RL method (R2D2). Solid lines denote average undiscounted task return; shaded regions denote standard deviation across 5 random seeds.

Figure 9.4 shows the performance of our integrated agent in two sparse-reward problems. We compare against Coin Flipping Networks (CFN) (Lobel et al., 2023), which is a state-of-the-art count-based exploration algorithm and R2D2 (Kapturowski et al., 2019), which is a distributed version of deep Q-learning (Watkins and Dayan, 1992; Mnih et al., 2015). These results corroborate our hypothesis that the drive to discover and use targeted options results in rapid exploration in challenging problems.

9.4 Conclusion

We presented an intrinsically motivated algorithm that learns reusable options that focus on achieving only certain aspects of state. The agent first explores the environment and when it finds a state that is particularly novel, it identifies the minimal set of features needed to explain the novelty of that state. Then it builds a classifier that focusses on only those features, while ignoring the rest. This classifier describes the subtask of a new option that is added to the agent's arsenal. We showed qualitatively how our discovered options differ from existing methods that focus on state-reaching. Finally, we showed that our HRL agent rapidly explores its environment and in doing so, effectively solves long-horizon RL problems.

Our agent suffers from at least two drawbacks. First, the methods used to extract the minimal set of relevant features exploits the simplicity of our test environments. Second, the policy over options $\pi_{\mathcal{O}}$ is model-free, and can be significantly improved by using high-level planning, similar to the IM-DSG agent discussed in Chapter 7.

CHAPTER 10

Discussion and Conclusion

The goal of this thesis was to address the question of abstraction discovery within the context of reinforcement learning. We sought to create agents capable of acquiring their own abstractions autonomously by interacting with the environment and in doing so, demonstrate effective exploration and planning.

In the pursuit of this goal, we made three concrete contributions. The first was on the topic of discovering skills that can be **reliably composed together in time**; this was done via the deep skill chaining algorithm and by addressing issues of non-stationarity that arise in the option discovery process. While this contribution focused only on action abstraction, our second contribution was to confront the challenges of **simultaneous acquisition of state and action abstractions** to create a new, abstract decision processes amenable to planning; this was first done with the deep skill graphs algorithm, which was later scaled using intrinsic motivations. Our final contribution was a framework for skill discovery that considered **exploration beyond coverage**. The Proto-goal RL agent finds controllable, reachable and desirable subspaces of the goal space for targeted exploration; this was finally extended by enabling the agent to ask counterfactual questions that probed the factored nature its observations.

However, each algorithm proposed in this thesis have several shortcomings that merit further research.

10.1 Limitations and Future Work

Deep skill chaining (DSC) improved upon non-hierarchical RL algorithms in long-horizon problems. It did so by learning options such that each option drove the agent to states inside the initiation region of another option. Methods to learn such initiation functions relied on a binary subtask description for the option, which limits the space of problems to which our method can be applied. Future work should consider ways to learn option initiation functions for arbitrary subtask specifications.

Furthermore, DSC chained together options that each control the full state of the agent. As discussed in Chapter 9, this is not realistic (or even desirable) in large, realistic domains. Methods that combine the factored nature of subgoal classifiers learned in Chapter 9 with chainability objectives will likely result in more powerful hierarchical agents.

The Intrinsically Motivated Deep Skill Graphs (IM-DSG) agent lifted some of the assumptions of DSG (state-sampler and distance function). But, it still has a special notion of an “expansion node”; it should be possible to construct an algorithm that more carefully trades-off exploration and exploitation in the abstract MDP, perhaps by taking inspiration from classical model-based RL works (Strehl and Littman, 2008). Furthermore, planning in the abstract MDP was done using value iteration, but asynchronous, incremental and approximate versions, or even hierarchical planners (Garrett et al., 2021), might be more scalable choices for future work.

Proto-goal RL conditions the agent’s value function on binary vectors, where each bit represents a subgoal achievement classifier of state. While each classifier exploits structure in the input space to make its decisions, this structure in the goal-space is largely unobservable to the agent's policy/value function. Future work should investigate ways in which each subgoal can be described using smooth representations that better prime the agent's goal-conditioned policy for stronger generalization.

On the topic of goal-conditioned policies, we considered two approaches in this thesis: one in which each option has its own function approximator, and second in which all options shared the same function approximator to enable generalization across options. Future work should investigate *intermediate* parameterizations that demonstrate generalization without incurring interference (Schaul et al., 2019).

Chapters 8 and 9 that developed proto-goal exploration, did not build an abstract model for planning. We wish to make more abstract models that are more expressive than the graphs of Chapters 6 and 7, perhaps in the style of Konidaris et al. (2018). Furthermore, future work could build multi-level hierarchies, perhaps taking inspiration from the *skill-symbol loop* (Konidaris, 2016).

All our exploration algorithms resort to a simplified strategy: first exploit and then explore. While this is commonplace, it is not always the correct strategy (Ecoffet et al., 2021). For example, in the video game Starcraft, more meaningful exploratory choices are made at the *beginning* of an episode, when the agent has to pick between different choices of weapons, armor, etc. In general, it should be up to the agent on when to explore more freely and when to carefully exploit using existing skills. The works of Pislar et al. (2022) and Bauer et al. (2023) could serve as starting points in this endeavor.

Finally, there are limits to how much a single agent may be able to discover about the environment. Richer forms of exploration include learning skills in socio-cultural settings (Sigaud et al., 2021), and collaboratively exploring large, complex environments in the multi-agent setting.

10.2 Closing Remarks

Exploration and discovery are among the most fascinating aspects of natural intelligence. Scientific study too can be viewed as a form of exploration: we ask questions about the natural world, and use the scientific method (alongside a collaborative spirit) to find answers to those questions (Bush, 1945). Not only do we possess the ability to find answers to questions, we also have an intriguing capability of finding good questions to ask in the first place. The option discovery problem, in its most general and powerful form, is akin to this spirit of exploration: find questions that are worth asking (cumulants) and figure out a way to answer them via trial-and-error (policies).

Understanding how humans and other animals learn to ask good questions may be one of the biggest (meta-) scientific problems of our time. This thesis reflects a smaller-than-minuscule step towards understanding the process of exploration and discovery by attempting to create artificial agents that posses such capabilities in small, toy, simulated environments. Ultimately, AI agents that can ask good questions and autonomously find answers to those questions may be the most scalable kind (Sutton, 2019); they will add to the wealth of human knowledge and in doing so, will dramatically expand our understanding of the natural world, and might qualitatively improve the lives of all.

References

Abel, D. (2020). *A Theory of Abstraction in Reinforcement Learning.* PhD thesis, Brown University.

Achiam, J., Edwards, H., Amodei, D., and Abbeel, P. (2018). Variational Option Discovery Algorithms. *CoRR*, abs/1807.10299.

Allen, C., Parikh, N., Gottesman, O., and Konidaris, G. (2021). Learning Markov State Abstractions for Deep Reinforcement Learning. *Advances in Neural Information Processing Systems*, 34:8229–8241.

Allen, C. S. (2023). *Structured Abstractions for General-Purpose Decision Making.* PhD thesis, Brown University.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. (2017). Hindsight Experience Replay. In *Advances in Neural Information Processing Systems*, pages 5048–5058.

Asadi, K., Misra, D., and Littman, M. L. (2018). Lipschitz Continuity in Model-Based Reinforcement Learning. In Dy, J. G. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 264–273. PMLR.

Auer, P. (2002). Using Confidence Bounds for Exploitation-Exploration Trade-Offs. *Journal of Machine Learning Research*, 3(Nov):397–422.

Auer, P., Jaksch, T., and Ortner, R. (2008). Near-Optimal Regret Bounds for Reinforcement Learning. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 21, Proceedings of the Twenty-Second Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 8-11, 2008*, pages 89–96. Curran Associates, Inc.

Bacon, P.-L., Harb, J., and Precup, D. (2017). The Option-Critic Architecture. In *Thirty-First AAAI Conference on Artificial Intelligence.*

Bacon, P.-L. and Precup, D. (2016). A Matrix Splitting Perspective on Planning with Options. *CoRR*, abs/1612.00916.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. (2020a). Agent57: Outperforming the Atari Human Benchmark. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 507–517. PMLR.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. (2020b). Never Give up: Learning Directed Exploration Strategies. In *International Conference on Learning Representations*. OpenReview.

Bagaria, A., Abbatematteo, B. M., Gottesman, O., Corsaro, M., Rammohan, S., and Konidaris, G. (2023). Effectively Learning Initiation Sets in Hierarchical Reinforcement Learning. In *Thirty-Seventh Conference on Neural Information Processing Systems (NeurIPS).*

Bagaria, A. and Konidaris, G. (2020a). Option Discovery Using Deep Skill Chaining. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.

Bagaria, A. and Konidaris, G. (2020b). Option Discovery Using Deep Skill Chaining. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.

Bagaria, A. and Schaul, T. (2023). Scaling Goal-Based Exploration via Pruning Proto-Goals. In *Proceedings of the Thirty-Second International Joint Conference on Artificial*

Intelligence, IJCAI 2023, 19th-25th August 2023, Macao, SAR, China, pages 3451–3460. IJCAI.

Bagaria, A., Senthil, J., Slivinski, M., and Konidaris, G. (2021a). Robustly Learning Composable Options in Deep Reinforcement Learning. In *30th International Joint Conference on Artificial Intelligence*. IJCAI.

Bagaria, A., Senthil, J. K., and Konidaris, G. (2021b). Skill Discovery for Exploration and Planning Using Deep Skill Graphs. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 521–531. PMLR.

Baranes, A. and Oudeyer, P.-Y. (2013). Active Learning of Inverse Models with Intrinsically Motivated Goal Exploration in Robots. *Robotics and Autonomous Systems*, 61(1):49–73.

Barreto, A., Borsa, D., Hou, S., Comanici, G., Aygün, E., Hamel, P., Toyama, D., Mourad, S., Silver, D., Precup, D., et al. (2019a). The Option Keyboard: Combining Skills in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 13031–13041.

Barreto, A., Borsa, D., Quan, J., Schaul, T., Silver, D., Hessel, M., Mankowitz, D. J., Zídek, A., and Munos, R. (2019b). Transfer in Deep Reinforcement Learning Using Successor Features and Generalised Policy Improvement. *CoRR*, abs/1901.10964.

Barreto, A., Dabney, W., Munos, R., Hunt, J. J., Schaul, T., van Hasselt, H. P., and Silver, D. (2017). Successor Features for Transfer in Reinforcement Learning. *Advances in neural information processing systems*, 30.

Barto, A. G. (2013). Intrinsic Motivation and Reinforcement Learning. In Baldassarre, G. and Mirolli, M., editors, *Intrinsically Motivated Learning in Natural and Artificial Systems*, pages 17–47. Springer.

Barto, A. G. and Mahadevan, S. (2003). Recent Advances in Hierarchical Reinforcement Learning. *Discrete event dynamic systems*, 13(1-2):41–77.

Barto, A. G., Singh, S., and Chentanez, N. (2004). Intrinsically Motivated Learning of Hierarchical Collections of Skills. In *Proceedings of the 3rd International Conference on Development and Learning*, pages 112–19.

Bauer, J., Baumli, K., Behbahani, F. M. P., Bhoopchand, A., Bradley-Schmieg, N., Chang, M., Clay, N., Collister, A., Dasagi, V., Gonzalez, L., Gregor, K., Hughes, E., Kashem, S., Loks-Thompson, M., Openshaw, H., Parker-Holder, J., Pathak, S., Nieves, N. P., Rakicevic, N., Rocktäschel, T., Schroecker, Y., Singh, S., Sygnowski, J., Tuyls, K., York, S., Zacherl, A., and Zhang, L. M. (2023). Human-Timescale Adaptation in an Open-Ended Task Space. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J., editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 1887–1935. PMLR.

Baumli, K., Warde-Farley, D., Hansen, S., and Mnih, V. (2021). Relative Variational Intrinsic Control. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*, pages 6732–6740.

Beck, J., Vuorio, R., Liu, E. Z., Xiong, Z., Zintgraf, L. M., Finn, C., and Whiteson, S. (2023). A Survey of Meta-Reinforcement Learning. *CoRR*, abs/2301.08028.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying Count-Based Exploration and Intrinsic Motivation. In *Advances in Neural Information Processing Systems*, pages 1471–1479.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. (2013). The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279.

Bellman, R. (1966). Dynamic Programming. *Science*, 153(3731):34–37.

Berseth, G., Geng, D., Devin, C. M., Rhinehart, N., Finn, C., Jayaraman, D., and Levine, S. (2021). SMiRL: Surprise Minimizing Reinforcement Learning in Unstable Environments. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.

Bertsekas, D. P. and Tsitsiklis, J. N. (1991). An Analysis of Stochastic Shortest Path Problems. *Mathematics of Operations Research*, 16(3):580–595.

Bishop, C. M. and Nasrabadi, N. M. (2006). *Pattern Recognition and Machine Learning*, volume 4. Springer.

Borsa, D., Barreto, A., Quan, J., Mankowitz, D. J., van Hasselt, H., Munos, R., Silver, D., and Schaul, T. (2019). Universal Successor Features Approximators. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Bradtke, S. J. and Duff, M. O. (1995). Reinforcement Learning Methods for Continuous-Time Markov Decision Problems. In *Advances in Neural Information Processing Systems*, pages 393–400.

Brafman, R. I. and Tennenholtz, M. (2002). R-Max - a General Polynomial Time Algorithm for near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3(Oct):213–231.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai Gym. *arXiv preprint arXiv:1606.01540*.

Brunskill, E. and Li, L. (2014). Pac-Inspired Option Discovery in Lifelong Reinforcement Learning. In *International Conference on Machine Learning*, pages 316–324.

Burda, Y., Edwards, H., Pathak, D., Storkey, A., Darrell, T., and Efros, A. A. (2019a).

Large-Scale Study of Curiosity-Driven Learning. In *International Conference on Learning Representations (ICLR)*. OpenReview.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2019b). Exploration by Random Network Distillation. In *International Conference on Learning Representations*.

Burridge, R. R., Rizzi, A. A., and Koditschek, D. E. (1999). Sequential Composition of Dynamically Dexterous Robot Behaviors. *The International Journal of Robotics Research*, 18(6):534–555.

Bush, V. (1945). *Science, the Endless Frontier: A Report to the President on a Program for Postwar Scientific Research*. United States Government Printing Office, Washington, D.C.

Campos Camúñez, V., Trott, A., Xiong, C., Socher, R., Giró Nieto, X., and Torres Viñals, J. (2020). Explore, Discover and Learn: Unsupervised Discovery of State-Covering Skills. In *Proceedings of the 37th International Conference on Machine Learning, ICML*, volume 119, pages 1317–1327.

Carvalho, W. T., Filos, A., Lewis, R., Lee, H., and Singh, S. (2022). Composing Task Knowledge with Modular Successor Feature Approximators. In *Deep Reinforcement Learning Workshop NeurIPS 2022*.

Chentanez, N., Barto, A. G., and Singh, S. P. (2005). Intrinsically Motivated Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 1281–1288.

Chevalier-Boisvert, M., Dai, B., Towers, M., de Lazcano, R., Willems, L., Lahlou, S., Pal, S., Castro, P. S., and Terry, J. (2023). Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. *CoRR*, abs/2306.13831.

Choi, J., Sharma, A., Lee, H., Levine, S., and Gu, S. S. (2021). Variational Empowerment as Representation Learning for Goal-Conditioned Reinforcement Learning. In *International Conference on Machine Learning*, pages 1953–1963. PMLR.

Chung, F. R. and Graham, F. C. (1997). *Spectral Graph Theory*. Number 92. American Mathematical Soc.

Colas, C., Karch, T., Sigaud, O., and Oudeyer, P.-Y. (2022). Autotelic Agents with Intrinsically Motivated Goal-Conditioned Reinforcement Learning: A Short Survey. *Journal of Artificial Intelligence Research*, 74:1159–1199.

Cortes, C. and Vapnik, V. (1995). Support-Vector Networks. *Machine learning*.

Dabney, W., Barreto, A., Rowland, M., Dadashi, R., Quan, J., Bellemare, M. G., and Silver, D. (2021). The Value-Improvement Path: Towards Better Representations for Reinforcement Learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, the Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 7160–7168. AAAI Press.

Dayan, P. (1993). Improving Generalization for Temporal Difference Learning: The Successor Representation. *Neural Computation*, 5(4):613–624.

Dayan, P. and Hinton, G. E. (1993). Feudal Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 271–278.

Deisenroth, M. and Rasmussen, C. E. (2011). PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *International Conference on Machine Learning*, pages 465–472.

Deng, L.-Y. (2006). The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation, and Machine Learning. *Technometrics : a journal of statistics for the physical, chemical, and engineering sciences*, 48(1):147–148.

Dietterich, T. G. (2000). Hierarchical Reinforcement Learning with the MAXQ Value Function Decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

Dijkstra, E. W. (1959). A Note on Two Problems in Connexion with Graphs. *Numerische mathematik*, 1(1):269–271.

Diuk, C., Cohen, A., and Littman, M. L. (2008). An Object-Oriented Representation for Efficient Reinforcement Learning. In *Proceedings of the 25th International Conference on Machine Learning*, Icml '08, pages 240–247, New York, NY, USA. Association for Computing Machinery.

Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., Sun, X., Xu, J., and Sui, Z. (2022). A Survey on In-Context Learning. *arXiv preprint arXiv:2301.00234*.

Du, Y., Kosoy, E., Dayan, A., Rufova, M., Abbeel, P., and Gopnik, A. (2023). What Can AI Learn from Human Exploration? Intrinsically-motivated Humans and Agents in Open-World Exploration. In *Neurips 2023 Workshop: Information-theoretic Principles in Cognitive Systems*.

Duan, Y., Chen, X., Houthooft, R., Schulman, J., and Abbeel, P. (2016). Benchmarking Deep Reinforcement Learning for Continuous Control. In *International Conference on Machine Learning*, pages 1329–1338.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2019). Go-Explore: A New Approach for Hard-Exploration Problems. *CoRR*, abs/1901.10995.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2021). First Return, Then Explore. *Nature*, 590(7847):580–586.

Eysenbach, B., Geng, X., Levine, S., and Salakhutdinov, R. (2020). Rewriting History with Inverse RL: Hindsight Inference for Policy Improvement. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.-F., and Lin, H.-T., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual*.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. (2019a). Diversity Is All You Need:

Learning Skills without a Reward Function. In *International Conference on Learning Representations*.

Eysenbach, B., Salakhutdinov, R. R., and Levine, S. (2019b). Search on the Replay Buffer: Bridging Planning and Reinforcement Learning. In *Advances in Neural Information Processing Systems 32*, pages 15246–15257.

Fang, K., Zhu, Y., Garg, A., Kurenkov, A., Mehta, V., Fei-Fei, L., and Savarese, S. (2020). Learning Task-Oriented Grasping for Tool Manipulation from Simulated Self-Supervision. *The International Journal of Robotics Research*, 39(2-3):202–216.

Farebrother, J., Greaves, J., Agarwal, R., Lan, C. L., Goroshin, R., Castro, P. S., and Bellemare, M. G. (2023). Proto-Value Networks: Scaling Representation Learning with Auxiliary Tasks. In *The Eleventh International Conference on Learning Representations*.

Forestier, S., Portelas, R., Mollard, Y., and Oudeyer, P.-Y. (2022). Intrinsically Motivated Goal Exploration Processes with Automatic Curriculum Learning. *Journal of Machine Learning Research (JMLR)*.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., Blundell, C., and Legg, S. (2018). Noisy Networks for Exploration. *International Conference on Learning Representations, ICLR*.

Frans, K., Ho, J., Chen, X., Abbeel, P., and Schulman, J. (2018). Meta Learning Shared Hierarchies. In *International Conference on Learning Representations*.

Fu, H., Yu, S., Tiwari, S., Littman, M., and Konidaris, G. (2023). Meta-Learning Parameterized Skills. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J., editors, *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 10461–10481. PMLR.

Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. (2020). D4RL: Datasets for

Deep Data-Driven Reinforcement Learning. *arXiv preprint arXiv:2004.07219*.

Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1582–1591.

Garcia, C. E., Prett, D. M., and Morari, M. (1989). Model Predictive Control: Theory and Practice—a Survey. *Automatica*, 25(3):335–348.

Garcia, J. and Fernández, F. (2012). Safe Exploration of State and Action Spaces in Reinforcement Learning. *Journal of Artificial Intelligence Research*, 45:515–564.

Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P., and Lozano-Pérez, T. (2021). Integrated Task and Motion Planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293.

Gershman, S. J. (2017). On the Blessing of Abstraction. *SAGE Publications Sage UK: London, England*.

Ghallab, M., Nau, D. S., and Traverso, P. (2016). *Automated Planning and Acting*. Cambridge University Press.

Ghavamzadeh, M., Lazaric, A., Maillard, O., and Munos, R. (2010). LSTD with Random Projections. *Advances in Neural Information Processing Systems*, 23.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT press.

Gopnik, A. (2020). Childhood as a Solution to Explore–Exploit Tensions. *Philosophical Transactions of the Royal Society B*, 375(1803):20190502.

Gopnik, A., Meltzoff, A. N., and Kuhl, P. K. (1999). *The Scientist in the Crib: Minds, Brains, and How Children Learn*. William Morrow & Co.

Gregor, K., Rezende, D. J., and Wierstra, D. (2016). Variational Intrinsic Control. *arXiv preprint arXiv:1611.07507*.

Gregor, K., Rezende, D. J., and Wierstra, D. (2017). Variational Intrinsic Control. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.

Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). Deep Reinforcement Learning for Robotic Manipulation with Asynchronous Off-Policy Updates. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396. IEEE.

Gupta, A., Kumar, V., Lynch, C., Levine, S., and Hausman, K. (2019). Relay Policy Learning: Solving Long-Horizon Tasks via Imitation and Reinforcement Learning. In Kaelbling, L. P., Kragic, D., and Sugiura, K., editors, *3rd Annual Conference on Robot Learning, CoRL 2019, Osaka, Japan, October 30 - November 1, 2019, Proceedings*, volume 100 of *Proceedings of Machine Learning Research*, pages 1025–1037. PMLR.

Gupta, A., Mendonca, R., Liu, Y., Abbeel, P., and Levine, S. (2018). Meta-Reinforcement Learning of Structured Exploration Strategies. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

Hafner, D., Lee, K.-H., Fischer, I., and Abbeel, P. (2022). Deep Hierarchical Planning from Pixels. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2018). Learning Latent Dynamics for Planning from Pixels. *arXiv preprint arXiv:1811.04551*.

Hafner, D., Ortega, P. A., Ba, J., Parr, T., Friston, K. J., and Heess, N. (2020). Action and Perception as Divergence Minimization. *CoRR*, abs/2009.01791.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. P. (2023). Mastering Diverse Domains through World Models. *CoRR*, abs/2301.04104.

Hamrick, J. B., Friesen, A. L., Behbahani, F., Guez, A., Viola, F., Witherspoon, S., Anthony, T., Buesing, L., Veličković, P., and Weber, T. (2020). On the Role of Planning in Model-Based Deep Reinforcement Learning. *arXiv preprint arXiv:2011.04021*.

Hansen, S., Dabney, W., Barreto, A., Warde-Farley, D., de Wiele, T. V., and Mnih, V. (2020). Fast Task Inference with Variational Intrinsic Successor Features. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.

Hansen, S., Desjardins, G., Baumli, K., Warde-Farley, D., Heess, N., Osindero, S., and Mnih, V. (2021). Entropic Desired Dynamics for Intrinsic Control. *Advances in Neural Information Processing Systems*, 34:11436–11448.

Harb, J., Bacon, P.-L., Klissarov, M., and Precup, D. (2018). When Waiting Is Not an Option: Learning Options with a Deliberation Cost. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Hartikainen, K., Geng, X., Haarnoja, T., and Levine, S. (2020). Dynamical Distance Learning for Semi-Supervised and Unsupervised Skill Discovery. In *International Conference on Learning Representations*.

Harutyunyan, A., Dabney, W., Borsa, D., Heess, N., Munos, R., and Precup, D. (2019). The Termination Critic. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2231–2240.

Hausknecht, M. J. and Stone, P. (2015). Deep Recurrent Q-Learning for Partially Observable Mdps. In *2015 AAAI Fall Symposia, Arlington, Virginia, USA, November 12-14, 2015*, pages 29–37. AAAI Press.

Heinlein, R. A. (1973). *Time Enough for Love*. Putnam Publishing Group.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining Improvements in

Deep Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework. In *International Conference on Learning Representations*.

Ho, M. K., Abel, D., Correa, C. G., Littman, M. L., Cohen, J. D., and Griffiths, T. L. (2021). People Construct Simplified Mental Representations to Plan. *Nature*, 606:129–136.

Huang, Z., Liu, F., and Su, H. (2019). Mapping State Space Using Landmarks for Universal Goal Reaching. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, pages 1940–1950.

Humplik, J., Galashov, A., Hasenclever, L., Ortega, P. A., Teh, Y. W., and Heess, N. (2019). Meta Reinforcement Learning as Task Inference. *CoRR*, abs/1905.06424.

Ivanov, A., Bagaria, A., and Konidaris, G. (2024). Discovering Options That Minimize Average Planning Time. In *The 39th Annual AAAI Conference on Artificial Intelligence*.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2017). Reinforcement Learning with Unsupervised Auxiliary Tasks. In *International Conference on Learning Representations*.

Jain, A., Khetarpal, K., and Precup, D. (2018). Safe Option-Critic: Learning Safety in the Option-Critic Architecture. *CoRR*, abs/1807.08060.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to Trust Your Model: Model-based Policy Optimization. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R., editors, *Advances in Neural Information*

Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada, pages 12498–12509.

Japkowicz, N. and Stephen, S. (2002). The Class Imbalance Problem: A Systematic Study. *Intelligent data analysis*, 6(5):429–449.

Jaques, N., Lazaridou, A., Hughes, E., Gülçehre, Ç., Ortega, P. A., Strouse, DJ., Leibo, J. Z., and de Freitas, N. (2019). Social Influence as Intrinsic Motivation for Multi-Agent Deep Reinforcement Learning. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 3040–3049. PMLR.

Javed, K. and Sutton, R. S. (2024). The Big World Hypothesis and Its Ramifications for Artificial Intelligence. In *Finding the Frame: An RLC Workshop for Examining Conceptual Frameworks*.

Jiang, M., Grefenstette, E., and Rocktäschel, T. (2021). Prioritized Level Replay. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 4940–4950. PMLR.

Jiang, N., Kulesza, A., Singh, S., and Lewis, R. L. (2016). The Dependence of Effective Planning Horizon on Model Accuracy. In Kambhampati, S., editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 4180–4189. IJCAI/AAAI Press.

Jinnai, Y., Abel, D., Hershkowitz, D., Littman, M., and Konidaris, G. (2018). Finding Options That Minimize Planning Time. In *Proceedings of the 36th International Conference on Machine Learning*.

Jinnai, Y., Park, J. W., Abel, D., and Konidaris, G. (2019). Discovering Options for

Exploration by Minimizing Cover Time. In *Proceedings of the 36th International Conference on Machine Learning*. PMLR.

Jinnai, Y., Park, J. W., Machado, M. C., and Konidaris, G. (2020). Exploration in Reinforcement Learning with Deep Covering Options. In *International Conference on Learning Representations*.

Jong, N. K., Hester, T., and Stone, P. (2008). The Utility of Temporal Abstraction in Reinforcement Learning. In *Aamas (1)*, pages 299–306.

Kaelbling, L. P. (1993). Learning to Achieve Goals. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1094–1099. Citeseer.

Kaelbling, L. P. and Lozano-Pérez, T. (2017). Learning Composable Models of Parameterized Skills. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 886–893. IEEE.

Kakade, S. M., Kearns, M. J., and Langford, J. (2003). Exploration in Metric State Spaces. In Fawcett, T. and Mishra, N., editors, *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003), August 21-24, 2003, Washington, DC, USA*, pages 306–312. AAAI Press.

Kaplan, F. and Oudeyer, P.-Y. (2004). Maximizing Learning Progress: An Internal Reward System for Development. In *Embodied Artificial Intelligence*, pages 259–270. Springer.

Kapturowski, S., Campos, V., Jiang, R., Rakićević, N., van Hasselt, H., Blundell, C., and Badia, A. P. (2022). Human-Level Atari 200x Faster. *arXiv preprint arXiv:2209.07550*.

Kapturowski, S., Ostrovski, G., Dabney, W., Quan, J., and Munos, R. (2019). Recurrent Experience Replay in Distributed Reinforcement Learning. In *International Conference on Learning Representations*.

Kavraki, L. E., Svestka, P., Latombe, J.-C., and Overmars, M. H. (1996). Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580.

Kearns, M. and Singh, S. (2002). Near-Optimal Reinforcement Learning in Polynomial Time. *Machine learning*, 49(2-3):209–232.

Khetarpal, K. and Precup, D. (2019). Learning Options with Interest Functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 9955–9956.

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In Bengio, Y. and LeCun, Y., editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*.

Klissarov, M., Bacon, P.-L., Harb, J., and Precup, D. (2017). Learnings Options End-to-End for Continuous Action Tasks. *Hierarchical Reinforcement Learning Workshop (NeurIPS)*.

Klissarov, M. and Machado, M. C. (2023). Deep Laplacian-Based Options for Temporally-Extended Exploration. *arXiv preprint arXiv:2301.11181*.

Klissarov, M. and Precup, D. (2021). Flexible Option Learning. In *Neural Information Processing Systems*.

Klyubin, A. S., Polani, D., and Nehaniv, C. L. (2005). Empowerment: A Universal Agent-Centric Measure of Control. In *2005 IEEE Congress on Evolutionary Computation*, volume 1, pages 128–135. IEEE.

Klyubin, A. S., Polani, D., and Nehaniv, C. L. (2008). Keep Your Options Open: An Information-Based Driving Principle for Sensorimotor Systems. *PloS one*, 3(12):e4018.

Knuth, D. E. and Moore, R. W. (1975). An Analysis of Alpha-Beta Pruning. *Artificial Intelligence*, 6(4):293–326.

Kokic, M., Kragic, D., and Bohg, J. (2020). Learning Task-Oriented Grasping from Human Activity Datasets. *IEEE Robotics and Automation Letters*, 5(2):3352–3359.

Kokic, M., Stork, J. A., Haustein, J. A., and Kragic, D. (2017). Affordance Detection for Task-Specific Grasping Using Deep Learning. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 91–98. IEEE.

Konidaris, G. (2019). On the Necessity of Abstraction. *Current Opinion in Behavioral Sciences*, 29:1–7.

Konidaris, G. and Barto, A. (2009a). Efficient Skill Learning Using Abstraction Selection. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Konidaris, G. and Barto, A. (2009b). Skill Discovery in Continuous Reinforcement Learning Domains Using Skill Chaining. In *Advances in Neural Information Processing Systems*, pages 1015–1023.

Konidaris, G., Kaelbling, L. P., and Lozano-Perez, T. (2018). From Skills to Symbols: Learning Symbolic Representations for Abstract High-Level Planning. *Journal of Artificial Intelligence Research*, 61:215–289.

Konidaris, G., Kuindersma, S., Grupen, R., and Barto, A. (2012). Robot Learning from Demonstration by Constructing Skill Trees. *The International Journal of Robotics Research*, 31(3):360–375.

Konidaris, G. D. (2016). Constructing Abstraction Hierarchies Using a Skill-Symbol Loop. In Kambhampati, S., editor, *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, pages 1648–1654. IJCAI/AAAI Press.

Konidaris, G. D. and Barto, A. G. (2007). Building Portable Options: Skill Transfer in Reinforcement Learning. In *20th International Joint Conference on Artificial Intelligence (IJCAI)*, volume 7, pages 895–900.

Kuffner, J. J. and LaValle, S. M. (2000). RRT-Connect: An Efficient Approach to Single-Query Path Planning. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation. Symposia Proceedings*, volume 2, pages 995–1001. IEEE.

Kumar, V. C., Ha, S., and Liu, C. K. (2018). Expanding Motor Skills Using Relay Networks. In *Conference on Robot Learning*, pages 744–756.

Küttler, H., Nardelli, N., Miller, A., Raileanu, R., Selvatici, M., Grefenstette, E., and Rocktäschel, T. (2020). The NetHack Learning Environment. *Advances in Neural Information Processing Systems*, 33:7671–7684.

Kwon, T. (2020). Variational Intrinsic Control Revisited. *CoRR*, abs/2010.03281.

Lagoudakis, M. G. and Parr, R. (2003). Least-Squares Policy Iteration. *The Journal of Machine Learning Research.*

Laskin, M., Liu, H., Peng, X. B., Yarats, D., Rajeswaran, A., and Abbeel, P. (2022). CIC: Contrastive Intrinsic Control for Unsupervised Skill Discovery.

Lattimore, T. and Szepesvári, C. (2020). *Bandit Algorithms.* Cambridge University Press, Cambridge.

LaValle, S. M. (1998). Rapidly-Exploring Random Trees: A New Tool for Path Planning.

Le, H., Voloshin, C., and Yue, Y. (2019-06-09/2019-06-15). Batch Policy Learning under Constraints. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3703–3712. PMLR.

Lee, S., Kim, J., Jang, I., and Kim, H. J. (2022). DHRL: A Graph-Based Approach for Long-Horizon and Sparse Hierarchical Reinforcement Learning. In Koyejo, S., Mohamed, S., Agarwal, A., Belgrave, D., Cho, K., and Oh, A., editors, *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022.*

Levy, A., Konidaris, G., Platt, R., and Saenko, K. (2019). Hierarchical Reinforcement Learning with Hindsight. In *International Conference on Learning Representations*.

Levy, A., Rammohan, S., Allievi, A., Niekum, S., and Konidaris, G. (2023). Hierarchical Empowerment: Towards Tractable Empowerment-Based Skill-Learning. *CoRR*, abs/2307.02728.

Li, S., Wang, R., Tang, M., and Zhang, C. (2019). Hierarchical Reinforcement Learning with Advantage-Based Auxiliary Rewards. In *Advances in Neural Information Processing Systems*, pages 1409–1419.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015). Continuous Control with Deep Reinforcement Learning. *arXiv preprint arXiv:1509.02971*.

Lin, L.-J. (1993). Reinforcement Learning for Robots Using Neural Networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science.

Lindemann, S. R. and LaValle, S. M. (2004). Incrementally Reducing Dispersion by Increasing Voronoi Bias in RRTs. In *IEEE International Conference on Robotics and Automation*, volume 4, pages 3251–3257.

Linke, C., Ady, N. M., White, M., Degris, T., and White, A. (2019). Adapting Behaviour via Intrinsic Reward: A Survey and Empirical Study. *CoRR*, abs/1906.07865.

Littman, M. (2017). The Reward Hypothesis. Technical report, Brown University. Video presentation can be found at https://tinyurl. com/4z52r3fe.

Littman, M. L., Dean, T. L., and Kaelbling, L. P. (1995). On the Complexity of Solving Markov Decision Problems. In *UAI '95: Proceedings of the Eleventh Annual Conference on Uncertainty in Artificial Intelligence, Montreal, Quebec, Canada, August 18-20, 1995*, pages 394–402. Morgan Kaufmann.

Liu, M., Zhu, M., and Zhang, W. (2022). Goal-Conditioned Reinforcement Learning: Problems and Solutions. *arXiv preprint arXiv:2201.08299*.

Lo, C., Mihucz, G., White, A., Aminmansour, F., and White, M. (2022). Goal-Space Planning with Subgoal Models. *CoRR*, abs/2206.02902.

Lo, C., Roice, K., Panahi, P. M., Jordan, S. M., White, A., Mihucz, G., Aminmansour, F., and White, M. (2024). Goal-Space Planning with Subgoal Models. *Journal of Machine Learning Research*, 25(330):1–57.

Lobel, S., Bagaria, A., and Konidaris, G. (2023). Flipping Coins to Estimate Pseudocounts for Exploration in Reinforcement Learning. In Krause, A., Brunskill, E., Cho, K., Engelhardt, B., Sabato, S., and Scarlett, J., editors, *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 22594–22613. PMLR.

Lobel, S., Gottesman, O., Allen, C., Bagaria, A., and Konidaris, G. (2022). Optimistic Initialization for Exploration in Continuous Control. In *Thirty-Sixth AAAI Conference on Artificial Intelligence, AAAI 2022, Thirty-Fourth Conference on Innovative Applications of Artificial Intelligence, IAAI 2022, The Twelveth Symposium on Educational Advances in Artificial Intelligence, EAAI 2022 Virtual Event, February 22 - March 1, 2022*, pages 7612–7619. AAAI Press.

Lopes, M., Lang, T., Toussaint, M., and Oudeyer, P.-Y. (2012). Exploration in Model-Based Reinforcement Learning by Empirically Estimating Learning Progress. *Advances in neural information processing systems*, 25.

Lowrey, K., Rajeswaran, A., Kakade, S., Todorov, E., and Mordatch, I. (2019). Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control. In *International Conference on Learning Representations*.

Lozano-Perez, T., Mason, M. T., and Taylor, R. H. (1984). Automatic Synthesis of

Fine-Motion Strategies for Robots. *The International Journal of Robotics Research*, 3(1):3–24.

Lyapunov, A. M. (1992). The General Problem of the Stability of Motion. *International journal of control*.

Lyle, C., Rowland, M., Ostrovski, G., and Dabney, W. (2021). On the Effect of Auxiliary Tasks on Representation Dynamics. In *International Conference on Artificial Intelligence and Statistics*, pages 1–9. PMLR.

Machado, M. C. (2019). *Efficient Exploration in Reinforcement Learning through Time-Based Representations*. PhD thesis, PhD thesis, University of Alberta, Canada.

Machado, M. C., Barreto, A., and Precup, D. (2023). Temporal Abstraction in Reinforcement Learning with the Successor Representation. *Journal of Machine Learning Research (JMLR)*, 24(80):1–69.

Machado, M. C., Bellemare, M. G., and Bowling, M. (2017). A Laplacian Framework for Option Discovery in Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, pages 2295–2304.

Machado, M. C., Bellemare, M. G., and Bowling, M. (2019). Count-Based Exploration with the Successor Representation.

Machado, M. C., Bellemare, M. G., Talvitie, E., Veness, J., Hausknecht, M., and Bowling, M. (2018a). Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research*, 61:523–562.

Machado, M. C., Rosenbaum, C., Guo, X., Liu, M., Tesauro, G., and Campbell, M. (2018b). Eigenoption Discovery through the Deep Successor Representation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.

Machado, M. C., Rosenbaum, C., Guo, X., Liu, M., Tesauro, G., and Campbell, M. (2018c). Eigenoption Discovery through the Deep Successor Representation. In *International Conference on Learning Representations*.

Mahadevan, S. and Maggioni, M. (2007). Proto-Value Functions: A Laplacian Framework for Learning Representation and Control in Markov Decision Processes. *Journal of Machine Learning Research*, 8(Oct):2169–2231.

Mandikal, P. and Grauman, K. (2021). Learning Dexterous Grasping with Object-Centric Visual Affordances. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6169–6176. IEEE.

Mandikal, P. and Grauman, K. (2022). Dexvip: Learning Dexterous Grasping with Human Hand Pose Priors from Video. In *Conference on Robot Learning*, pages 651–661. PMLR.

Mania, H., Guy, A., and Recht, B. (2018). Simple Random Search of Static Linear Policies Is Competitive for Reinforcement Learning. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 1805–1814.

Mann, T. A., Mannor, S., and Precup, D. (2015). Approximate Value Iteration with Temporally Extended Actions. *Journal of Artificial Intelligence Research*, 53:375–438.

Mataric, M. J. (1994). Reward Functions for Accelerated Learning. In *Machine Learning Proceedings 1994*, pages 181–189. Elsevier.

McCarthy, J. (1997). What Is Artificial Intelligence.

McClinton, W., Levy, A., and Konidaris, G. (2021). HAC Explore: Accelerating Exploration with Hierarchical Reinforcement Learning. *CoRR*, abs/2108.05872.

Mcgovern, E. A. (2002). *Autonomous Discovery of Temporal Abstractions from Interaction with an Environment*. PhD thesis, University of Massachusetts at Amherst.

Mendonca, R., Rybkin, O., Daniilidis, K., Hafner, D., and Pathak, D. (2021). Discovering and Achieving Goals via World Models. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual*, pages 24379–24391.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-Level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529.

Mohamed, S. and Jimenez Rezende, D. (2015). Variational Information Maximisation for Intrinsically Motivated Reinforcement Learning. *Advances in neural information processing systems*, 28.

Nachum, O., Gu, S., Lee, H., and Levine, S. (2019). Near-Optimal Representation Learning for Hierarchical Reinforcement Learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.

Nachum, O., Gu, S. S., Lee, H., and Levine, S. (2018). Data-Efficient Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 3303–3313.

Nagabandi, A., Kahn, G., Fearing, R. S., and Levine, S. (2018). Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE.

Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. (2018). Visual Reinforcement Learning with Imagined Goals. *Advances in Neural Information Processing Systems*, 31:9191–9200.

Nam, T., Sun, S.-H., Pertsch, K., Hwang, S. J., and Lim, J. J. (2022). Skill-Based Meta-Reinforcement Learning. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.

Nasiriany, S., Pong, V., Lin, S., and Levine, S. (2019). Planning with Goal-Conditioned Policies. *Advances in Neural Information Processing Systems (NeurIPS).*

Natarajan, N., Dhillon, I. S., Ravikumar, P. K., and Tewari, A. (2013). Learning with Noisy Labels. *Advances in neural information processing systems*, 26.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy Invariance under Reward Transformations: Theory and Application to Reward Shaping. In *International Conference on Machine Learning*, volume 99, pages 278–287.

Niekum, S. and Barto, A. G. (2011). Clustering via Dirichlet Process Mixture Models for Portable Skill Discovery. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 1818–1826. Curran Associates, Inc.

OEL Team, O. E. L. T., Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., McAleese, N., Bradley-Schmieg, N., Wong, N., Porcel, N., Raileanu, R., Hughes-Fitt, S., Dalibard, V., and Czarnecki, W. M. (2021). Open-Ended Learning Leads to Generally Capable Agents. *CoRR*, abs/2107.12808.

Oh, J., Hessel, M., Czarnecki, W. M., Xu, Z., van Hasselt, H. P., Singh, S., and Silver, D. (2020). Discovering Reinforcement Learning Algorithms. *Advances in Neural Information Processing Systems*, 33:1060–1070.

Osband, I., Aslanides, J., and Cassirer, A. (2018). Randomized Prior Functions for Deep Reinforcement Learning. *Advances in Neural Information Processing Systems*, 31.

Osband, I., Blundell, C., Pritzel, A., and Roy, B. V. (2016a). Deep Exploration via Bootstrapped DQN. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R., editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4026–4034.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. (2016b). Deep Exploration via Bootstrapped DQN. *Advances in neural information processing systems*, 29.

Osband, I., Russo, D., and Roy, B. V. (2013). (More) Efficient Reinforcement Learning via Posterior Sampling. In Burges, C. J. C., Bottou, L., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a Meeting Held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3003–3011.

Ostrovski, G., Bellemare, M. G., Oord, A., and Munos, R. (2017). Count-Based Exploration with Neural Density Models. In *International Conference on Machine Learning*, pages 2721–2730. PMLR.

Ostrovski, G., Castro, P. S., and Dabney, W. (2021). The Difficulty of Passive Learning in Deep Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 34, pages 23283–23295.

Packer, C., Gao, K., Kos, J., Krähenbühl, P., Koltun, V., and Song, D. (2018). Assessing Generalization in Deep Reinforcement Learning. *arXiv preprint arXiv:1810.12282*.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-Driven Exploration by Self-Supervised Prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 16–17.

Pislar, M., Szepesvari, D., Ostrovski, G., Borsa, D. L., and Schaul, T. (2022). When Should Agents Explore? In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.

Pitis, S., Chan, H., Jamali, K., and Ba, J. (2020a). An Inductive Bias for Distances: Neural Nets That Respect the Triangle Inequality. In *8th International Conference on Learning Representations, ICLR*.

Pitis, S., Chan, H., Zhao, S., Stadie, B., and Ba, J. (2020b). Maximum Entropy Gain

Exploration for Long Horizon Multi-Goal Reinforcement Learning. *arXiv preprint arXiv:2007.02832.*

Pitis, S., Chan, H., Zhao, S., Stadie, B. C., and Ba, J. (2020c). Maximum Entropy Gain Exploration for Long Horizon Multi-Goal Reinforcement Learning. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7750–7761. PMLR.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. (2018). Parameter Space Noise for Exploration. *International Conference on Learning Representations, ICLR.*

Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S. (2019). Skew-Fit: State-covering Self-Supervised Reinforcement Learning. *Proceedings of the 37th International Conference on Machine Learning, ICML.*

Precup, D. (2001). *Temporal Abstraction in Reinforcement Learning.* PhD thesis, University of Massachusetts at Amherst.

Precup, D., Sutton, R. S., and Singh, S. (1998). Theoretical Results on Reinforcement Learning with Temporally Abstract Options. In *Machine Learning: ECML-98: 10th European Conference on Machine Learning Chemnitz, Germany, April 21–23, 1998 Proceedings 10*, pages 382–393. Springer.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley Series in Probability and Statistics. Wiley.

Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. (2021). Learning Transferable Visual Models from Natural Language Supervision. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning,* *ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 8748–8763. PMLR.

Raileanu, R. and Rocktäschel, T. (2020). RIDE: Rewarding Impact-Driven Exploration for Procedurally-Generated Environments. In *International Conference on Learning Representations*.

Randløv, J. and Alstrøm, P. (1998). Learning to Drive a Bicycle Using Reinforcement Learning and Shaping. In *Proceedings of the 15th International Conference on Machine Learning*, volume 98, pages 463–471.

Randløv, J., Barto, A. G., and Rosenstein, M. T. (2000). Combining Reinforcement Learning with a Local Control Algorithm. In *International Conference on Machine Learning*, pages 775–782.

Raparthy, S. C., Hambro, E., Kirk, R., Henaff, M., and Raileanu, R. (2023). Generalization to New Sequential Decision Making Tasks with In-Context Learning. *arXiv preprint arXiv:2312.03801*.

Ring, M. (1994). *Continual Learning in Reinforcement Environments*. PhD thesis, University of Texas at Austin.

Röder, F., Eppe, M., Nguyen, P. D. H., and Wermter, S. (2020). Curious Hierarchical Actor-Critic Reinforcement Learning. In Farkas, I., Masulli, P., and Wermter, S., editors, *Artificial Neural Networks and Machine Learning - ICANN 2020 - 29th International Conference on Artificial Neural Networks, Bratislava, Slovakia, September 15-18, 2020, Proceedings, Part II*, volume 12397 of *Lecture Notes in Computer Science*, pages 408–419. Springer.

Roice, K., Panahi, P. M., Jordan, S. M., White, A., and White, M. (2024). A New View on Planning in Online Reinforcement Learning.

Rosen, E., Abbatematteo, B. M., Thompson, S., Akbulut, T., and Konidaris, G. (2022).

On the Role of Structure in Manipulation Skill Learning. In *CoRL 2022 Workshop on Learning, Perception, and Abstraction for Long-Horizon Planning*.

Russell, S. and Norvig, P. (2020). *Artificial Intelligence: A Modern Approach (4th Edition)*. Pearson.

Russell, S. J. and Subramanian, D. (1994). Provably Bounded-Optimal Agents. *Journal of Artificial Intelligence Research*, 2:575–609.

Samvelyan, M., Kirk, R., Kurin, V., Parker-Holder, J., Jiang, M., Hambro, E., Petroni, F., Küttler, H., Grefenstette, E., and Rocktäschel, T. (2021). MiniHack the Planet: A Sandbox for Open-Ended Reinforcement Learning Research. In *NeurIPS Datasets and Benchmarks Track*.

Savinov, N., Dosovitskiy, A., and Koltun, V. (2018). Semi-Parametric Topological Memory for Navigation. In *International Conference on Learning Representations*.

Schaul, T., Borsa, D., Modayil, J., and Pascanu, R. (2019). Ray Interference: A Source of Plateaus in Deep Reinforcement Learning. *arXiv preprint arXiv:1904.11455*.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal Value Function Approximators. In *International Conference on Machine Learning*, pages 1312–1320.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized Experience Replay. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

Schaul, T. and Ring, M. B. (2013). Better Generalization with Forecasts. In Rossi, F., editor, *IJCAI 2013, Proceedings of the 23rd International Joint Conference on Artificial Intelligence, Beijing, China, August 3-9, 2013*, pages 1656–1662. IJCAI/AAAI.

Schiavi, G., Wulkop, P., Rizzi, G., Ott, L., Siegwart, R., and Chung, J. J. (2022). Learning

Agent-Aware Affordances for Closed-Loop Interaction with Articulated Objects. *arXiv preprint arXiv:2209.05802*.

Schmidhuber, J. (1991). Curious Model-Building Control Systems. In *Proc. International Joint Conference on Neural Networks*, pages 1458–1463.

Schmidhuber, J. (2010a). Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010). *IEEE transactions on autonomous mental development*, 2(3):230–247.

Schmidhuber, J. (2010b). Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990-2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247.

Schrader, M.-P. B. (2018). Gym-Sokoban.

Schrittwieser, J., Hubert, T., Mandhane, A., Barekatain, M., Antonoglou, I., and Silver, D. (2021). Online and Offline Reinforcement Learning by Planning with a Learned Model. *Advances in Neural Information Processing Systems*, 34:27580–27591.

Shah, N. and Srivastava, S. (2024). Hierarchical Planning and Learning for Robots in Stochastic Settings Using Zero-Shot Option Invention. In Wooldridge, M. J., Dy, J. G., and Natarajan, S., editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pages 10358–10367. AAAI Press.

Sharma, A., Ahn, M., Levine, S., Kumar, V., Hausman, K., and Gu, S. (2020a). Emergent Real-World Robotic Skills via Unsupervised off-Policy Reinforcement Learning. In Toussaint, M., Bicchi, A., and Hermans, T., editors, *Robotics: Science and Systems XVI, Virtual Event / Corvalis, Oregon, USA, July 12-16, 2020*.

Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. (2020b). Dynamics-Aware Unsupervised Discovery of Skills. (arXiv:1907.01657).

Sharma, A., Gu, S., Levine, S., Kumar, V., and Hausman, K. (2020c). Dynamics-Aware Unsupervised Discovery of Skills. In *International Conference on Learning Representations (ICLR)*.

Shoeleh, F. and Asadpour, M. (2017). Graph Based Skill Acquisition and Transfer Learning for Continuous Reinforcement Learning Domains. *Pattern Recognition Letters*, 87:104–116.

Sigaud, O., Caselles-Dupré, H., Colas, C., Akakzia, A., Oudeyer, P.-Y., and Chetouani, M. (2021). Towards Teachable Autonomous Agents. *CoRR*, abs/2105.11977.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T. P., Simonyan, K., and Hassabis, D. (2017a). Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *CoRR*, abs/1712.01815.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017b). Mastering the Game of Go without Human Knowledge. *Nature*, 550(7676):354–359.

Silver, D., Singh, S., Precup, D., and Sutton, R. S. (2021). Reward Is Enough. *Artificial Intelligence*, 299:103535.

Simon, H. A. (1991). Bounded Rationality and Organizational Learning. *Organization science*, 2(1):125–134.

Şimşek, Ö. and Barto, A. G. (2004). Using Relative Novelty to Identify Useful Temporal Abstractions in Reinforcement Learning. In Brodley, C. E., editor, *Machine Learning, Proceedings of the Twenty-First International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, volume 69 of *ACM International Conference Proceeding Series*. ACM.

Şimşek, Ö. and Barto, A. G. (2006). An Intrinsic Reward Mechanism for Efficient

Exploration. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 833–840.

Şimşek, Ö., Wolfe, A. P., and Barto, A. G. (2005). Identifying Useful Subgoals in Reinforcement Learning by Local Graph Partitioning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 816–823. ACM.

Singh, A., Yu, A., Yang, J., Zhang, J., Kumar, A., and Levine, S. (2020). COG: Connecting New Skills to Past Experience with Offline Reinforcement Learning. In *Conference on Robot Learning*.

Smith, M., van Hoof, H., and Pineau, J. (2018-07-10/2018-07-15). An Inference-Based Policy Gradient Method for Learning Options. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4703–4712, Stockholmsmässan, Stockholm Sweden. PMLR.

Solway, A., Diuk, C., Córdova, N., Yee, D., Barto, A. G., Niv, Y., and Botvinick, M. M. (2014). Optimal Behavioral Hierarchy. *PLOS Computational Biology*, 10(8):1–10.

Stachenfeld, K. L., Botvinick, M. M., and Gershman, S. J. (2017). The Hippocampus as a Predictive Map. *Nature neuroscience*, 20(11):1643–1653.

Stout, A. and Barto, A. G. (2010). Competence Progress Intrinsic Motivation. In *2010 IEEE 9th International Conference on Development and Learning*, pages 257–262. IEEE.

Strehl, A. L. and Littman, M. L. (2008). An Analysis of Model-Based Interval Estimation for Markov Decision Processes. *Journal of Computer and System Sciences*, 74(8):1309–1331.

Strens, M. J. A. (2000). A Bayesian Framework for Reinforcement Learning. In Langley, P., editor, *Proceedings of the Seventeenth International Conference on Machine Learning*

(ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000, pages 943–950. Morgan Kaufmann.

Strouse, DJ., Baumli, K., Warde-Farley, D., Mnih, V., and Hansen, S. S. (2022). Learning More Skills through Optimistic Exploration. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.

Su, J., Liu, N., Wang, Y., Tenenbaum, J. B., and Du, Y. (2024). Compositional Image Decomposition with Diffusion Models. *CoRR*, abs/2406.19298.

Sutton, R. S. (1988). Learning to Predict by the Methods of Temporal Differences. *Machine Learning*, 3:9–44.

Sutton, R. S. (1990). Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming. In Porter, B. W. and Mooney, R. J., editors, *Machine Learning, Proceedings of the Seventh International Conference on Machine Learning, Austin, Texas, USA, June 21-23, 1990*, pages 216–224. Morgan Kaufmann.

Sutton, R. S. (1991). Dyna, an Integrated Architecture for Learning, Planning, and Reacting. *SIGART Bull.*, 2(4):160–163.

Sutton, R. S. (2004). The Reward Hypothesis. Technical report, University of Alberta. Available at http://incompleteideas.net/rlai.cs.ualberta.ca/RLAI/rewardhypothesis.html.

Sutton, R. S. (2016). The Future of Artificial Intelligence Belongs to Search and Learning. Technical report, University of Alberta. Slides available at http://www.incompleteideas.net/Talks/toronto2016.pdf.

Sutton, R. S. (2019). The Bitter Lesson. Technical report, University of Alberta. Available at http://www.incompleteideas.net/IncIdeas/BitterLesson.html.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* MIT press.

Sutton, R. S., Bowling, M., and Pilarski, P. M. (2022). The Alberta Plan for AI Research. *arXiv preprint arXiv:2208.11173*.

Sutton, R. S., Koop, A., and Silver, D. (2007). On the Role of Tracking in Stationary Environments. In *International Conference on Machine Learning*.

Sutton, R. S., Machado, M. C., Holland, G. Z., Szepesvari, D., Timbers, F., Tanner, B., and White, A. (2024). Reward-Respecting Subtasks for Model-Based Reinforcement Learning. In Wooldridge, M. J., Dy, J. G., and Natarajan, S., editors, *Thirty-Eighth AAAI Conference on Artificial Intelligence*, page 22713. AAAI Press.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. (2011). Horde: A Scalable Real-Time Architecture for Learning Knowledge from Unsupervised Sensorimotor Interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768. International Foundation for Autonomous Agents and Multiagent Systems.

Sutton, R. S., Precup, D., and Singh, S. (1999). Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning. *Artificial Intelligence*, 112(1):181–211.

Suzuki, S. and Abe, K. (1985). Topological Structural Analysis of Digitized Binary Images by Border Following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46.

Taïga, A. A., Fedus, W., Machado, M. C., Courville, A., and Bellemare, M. G. (2019). Benchmarking Bonus-Based Exploration Methods on the Arcade Learning Environment. *arXiv preprint arXiv:1908.02388*.

Talvitie, E. (2014). Model Regularization for Stable Sample Rollouts. In Zhang, N. L. and Tian, J., editors, *Proceedings of the Thirtieth Conference on Uncertainty in Artificial Intelligence, UAI 2014, Quebec City, Quebec, Canada, July 23-27, 2014*, pages 780–789. AUAI Press.

Talvitie, E. (2017). Self-Correcting Models for Model-Based Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Tasse, G. N., James, S., and Rosman, B. (2021). Generalisation in Lifelong Reinforcement Learning through Logical Composition. In *International Conference on Learning Representations*.

Tax, D. M. and Duin, R. P. (1999). Support Vector Domain Description. *Pattern recognition letters*.

Taylor, M. E. and Stone, P. (2009). Transfer Learning for Reinforcement Learning Domains: A Survey. *Journal of Machine Learning Research*, 10(7).

Tedrake, R. (2009). LQR-trees: Feedback Motion Planning on Sparse Randomized Trees.

Teikari, A. (2019). Baba Is You. *Game [PC], Hempuli Oy, Finland*.

Tenenbaum, J. (2018). Building Machines That Learn and Think like People. In André, E., Koenig, S., Dastani, M., and Sukthankar, G., editors, *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, page 5. International Foundation for Autonomous Agents and Multiagent Systems Richland, SC, USA / ACM.

Tenenbaum, J. B., Kemp, C., Griffiths, T. L., and Goodman, N. D. (2011). How to Grow a Mind: Statistics, Structure, and Abstraction. *science*, 331(6022):1279–1285.

Thompson, W. R. (1933). On the Likelihood That One Unknown Probability Exceeds Another in View of the Evidence of Two Samples. *Biometrika*, 25(3-4):285–294.

Tiwari, S. and Thomas, P. S. (2019). Natural Option Critic. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 5175–5182.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A Physics Engine for Model-Based Control. In *International Conference on Intelligent Robots and Systems*, pages 5026–5033.

Touati, A. and Ollivier, Y. (2021). Learning One Representation to Optimize All Rewards. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual*, pages 13–23.

van den Oord, A., Li, Y., and Vinyals, O. (2018). Representation Learning with Contrastive Predictive Coding. *CoRR*, abs/1807.03748.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. In Schuurmans, D. and Wellman, M. P., editors, *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*, pages 2094–2100. AAAI Press.

Veeriah, V., Hessel, M., Xu, Z., Rajendran, J., Lewis, R. L., Oh, J., van Hasselt, H. P., Silver, D., and Singh, S. (2019). Discovery of Useful Questions as Auxiliary Tasks. *Advances in Neural Information Processing Systems*, 32.

Veeriah, V., Oh, J., and Singh, S. (2018). Many-Goals Reinforcement Learning. *arXiv preprint arXiv:1806.09605*.

Veeriah, V., Zahavy, T., Hessel, M., Xu, Z., Oh, J., Kemaev, I., van Hasselt, H., Silver, D., and Singh, S. (2021). Discovery of Options via Meta-Learned Subgoals. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, Virtual*, pages 29861–29873.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. (2017). Feudal Networks for Hierarchical Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3540–3549.

Voloshin, C., Le, H. M., Jiang, N., and Yue, Y. (2021). Empirical Study of Off-Policy Policy Evaluation for Reinforcement Learning. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, Virtual.*

Volpi, N. C. and Polani, D. (2023). Goal-Directed Empowerment: Combining Intrinsic Motivation and Task-Oriented Behavior. *IEEE Trans. Cogn. Dev. Syst.*, 15(2):361–372.

Wan, Y. and Sutton, R. S. (2022). Toward Discovering Options That Achieve Faster Planning. *CoRR*, abs/2205.12515.

Wang, K., Zhou, K., Zhang, Q., Shao, J., Hooi, B., and Feng, J. (2021). Towards Better Laplacian Representation in Reinforcement Learning with Generalized Graph Drawing. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 11003–11012. PMLR.

Warde-Farley, D., de Wiele, T. V., Kulkarni, T. D., Ionescu, C., Hansen, S., and Mnih, V. (2019a). Unsupervised Control through Non-Parametric Discriminative Rewards. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.* OpenReview.

Warde-Farley, D., de Wiele, T. V., Kulkarni, T. D., Ionescu, C., Hansen, S., and Mnih, V. (2019b). Unsupervised Control through Non-Parametric Discriminative Rewards. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.* OpenReview.

Watkins, C. J. and Dayan, P. (1992). Q-Learning. *Machine learning*, 8(3-4):279–292.

Wen, B., Lian, W., Bekris, K., and Schaal, S. (2022). Catgrasp: Learning Category-Level Task-Relevant Grasping in Clutter from Simulation. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 6401–6408. IEEE.

Wen, Z., Precup, D., Ibrahimi, M., Barreto, A., Van Roy, B., and Singh, S. (2020). On Efficiency in Hierarchical Reinforcement Learning. In *Advances in Neural Information Processing Systems*, volume 33, pages 6708–6718.

White, A. (2015). *Developing a Predictive Approach to Knowledge*. PhD thesis, University of Alberta.

White, M. (2017). Unifying Task Specification in Reinforcement Learning. In *International Conference on Machine Learning*, pages 3742–3750. PMLR.

White, R. W. (1959). Motivation Reconsidered: The Concept of Competence. *Psychological review*, 66(5):297.

Williams, G., Drews, P., Goldfain, B., Rehg, J. M., and Theodorou, E. A. (2016). Aggressive Driving with Model Predictive Path Integral Control. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1433–1440.

Wu, Y., Tucker, G., and Nachum, O. (2019). The Laplacian in RL: Learning Representations with Efficient Approximations. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.

Xu, K., Verma, S., Finn, C., and Levine, S. (2020). Continual Learning of Control Primitives : Skill Discovery via Reset-Games. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M.-F., and Lin, H.-T., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, Virtual.*

Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2020). Meta-World: A Benchmark and Evaluation for Multi-Task and Meta Reinforcement Learning. In *Conference on Robot Learning (CoRL)*, volume 100, pages 1094–1100.

Zhang, J., Yu, H., and Xu, W. (2021a). Hierarchical Reinforcement Learning by Discovering Intrinsic Options. In *International Conference on Learning Representations.*

Zhang, L., Yang, G., and Stadie, B. C. (2021b). World Model as a Graph: Learning Latent Landmarks for Planning. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 12611–12620. PMLR.

Zhang, S. and Whiteson, S. (2019). DAC: The Double Actor-Critic Architecture for Learning Options. In *Advances in Neural Information Processing Systems (NeurIPS).*

Zhao, J., Troniak, D., and Kroemer, O. (2021). Towards Robotic Assembly by Predicting Robust, Precise and Task-Oriented Grasps. In *Conference on Robot Learning*, pages 1184–1194. PMLR.

Zhu, Y., Wong, J., Mandlekar, A., Martín-Martín, R., Joshi, A., Nasiriany, S., and Zhu, Y. (2020). Robosuite: A Modular Simulation Framework and Benchmark for Robot Learning. *arXiv preprint arXiv:2009.12293.*