ABSTRACT

Dissertation title: Abstract Decision Making and Concept Formation for Adaptability and Generalization

John Winder
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County

Co-directed by: Cynthia Matuszek
Assistant Professor
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County

and

Marie desJardins
Dean
College of Organizational, Computational, and Information Sciences
Simmons University

Generalization remains a central challenge for machine learning algorithms, especially when embodied in artificially intelligent agents that learn and plan under uncertainty. By using reinforcement learning (RL) or probabilistic planning techniques, such agents may be trained successfully to excel at solving a specific, narrow task. Upon transfer to a different environment, however, where they face novelty in the form of new goals or unusual surroundings, their lack of an ability to adapt is most clearly highlighted by degraded performance. In contrast, humans possess a facility for adaptation. We create and recall concepts that enable us to interpret any anomalies we encounter. Likewise, we develop and repeat habits that help us navigate our life, allowing us to think further into

the future by alleviating the burden of contemplating all the details of tasks we tackle in a common day.

In this thesis, I aim to make agents more adaptable by developing new methods for reasoning abstractly. Through a process of concept formation, agents expand their understanding of entities in the world such that any anomalies may be interpreted based on their conceptual relation to what has already been learned. I develop an algorithm for concept-aware feature extraction, such that agents maintain a conceptual knowledge base that grows to accommodate new concepts. Exploring the application of this approach to two decision-making paradigms—contextual bandits and temporal difference RL—I demonstrate how explicitly reasoning about concepts makes agents adapt more readily when facing a stream of anomalous objects or upon transfer to harder tasks.

For habits, I articulate how decision-making agents may assemble useful patterns of behavior into formal structures called subtasks, which aid an agent's ability to reason abstractly, over varying timescales. Subtasks, thus, facilitate creating and reusing solutions to common problems. I build upon two separate formulations of subtasks: the options framework (a standard approach to hierarchical RL) and abstract Markov decision processes (AMDPs). I develop new algorithms to investigate how abstract option models may be approximated efficiently from experience, how abstract option policies may be adapted to novel tasks, and how hierarchies of AMDPs let agents plan more flexibly and effectively at varying levels of abstraction. Finally, I combine these ideas to make a new model-based RL algorithm for planning with abstract, learned models: an agent creates AMDP subtasks bottom-up from data and learns to plan with them top-down, using the hierarchy it generated to generalize and adapt to variant tasks.

Abstract Decision Making and Concept Formation for Adaptability and Generalization

by

John Winder

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2019

Advisory Committee:
Professor Cynthia Matuszek, Chair, Co-Advisor
Professor Marie desJardins, Co-Advisor
Professor Tim Finin
Professor Tim Oates
Professor Stefanie Tellex

Acknowledgments

Before one begins a dissertation, there is really no way to understand just how many people become intertwined in the endeavor. Reflecting on the myriad ways my work has relied upon and benefited from engaging with those around me, I feel truly humbled. With deepest thanks, it is my pleasure to say I have been graced with excellent advisors, collaborators, colleagues, students, and friends. I hope to acknowledge them here and offer my appreciation for our collaboration.

First, I would like to give my deepest thanks to Marie desJardins, the advisor through not only my doctoral but undergraduate career as well. In reflecting on our work together, I recall the first time we met was in Fall 2009, my first semester of college. She was presenting at the Honors College forum, for which professors were invited from around UMBC to give a talk on their work, and touch upon their vision of the future. Marie, of course, spoke about artificial intelligence—I was enthralled, inquisitive, and compelled to learn more. Now this lecture was prior to my decision to pursue computer science, and in hindsight played a key role in my choices thereafter. I took the first opportunity to enroll in one of Marie's classes, her popular Complexity and Emergence seminar. As her student, Marie graciously invited me to attend MAPLE lab sessions and become involved in academic research as an undergraduate, an uncommon and valuable privilege. Due to this experience, in which Marie encouraged my curiosity while influencing how I should view research, I also became interested in computer science education. For my graduate program, I continued my artificial intelligence work with Marie, focusing dually on the academic research that led ultimately to this document, but also contributing to the

CS Matters in Maryland project. This effort aimed, and succeeded, in getting high quality computer science instruction, materials, and training to high school teachers across not only Maryland, but the whole country (and even a few internationally!). It was under Marie's counsel and tutelage that I enjoyed both this breadth of experience as well as a depth of work I hope is reflected herein. The foundation of a great relationship between a student and mentor is built upon honesty and trust, which is especially true of ours; I cannot overstate my gratitude for it.

Secondly, I am also happy to thank Cynthia Matuszek, who has become my second advisor during this final year. Cynthia's insight and attention has helped improve and shape the quality of my writing and thought, which has been especially critical in these past few months dedicated to the completion of this dissertation. I highly value all the advice and guidance she has provided to help me reach this point. Similarly, I would like to thank Tim Oates who co-advised me during my Master's research. Under his advice during and since that time, I have become enlightened on the process of research, how to structure experiments, and organize the empirical analysis of an investigation. Altogether, I have been exceptionally fortunate in having intelligent, thoughtful, and motivating advisors.

In addition, I would like to extend my appreciation to the rest of my committee. In various conversations with Tim Finin, I gained better understanding of what areas to pursue in terms of structured knowledge, what opportunities are available in terms of research, and the broader picture of my academic work. Stefanie Tellex has helped challenge the boundaries of where my work can go, offering trenchant feedback on what problems to consider, how to assess the motivation, impact, and long-term vision of the

work. I have also been fortunate to cooperate on the AMDP project with her and her students in the H2R lab. Though not a member of my committee, I would also like to thank another collaborator, Michael Littman, whose enthusiasm and depth of thought has been a constant source of inspiration. Finally, Frank Ferraro offered helpful feedback on my presentations and writing, for which I am much appreciative.

Other colleagues who hold my respect and regard include my fellow students and postdoctoral co-authors on various papers and projects. At UMBC: Karan Budhraja, Jon Clancy, Nicholas Haltmeyer, Matthew Landen, James MacGlashan, Stephanie Milani, Erebus Oh, Shane Parr, Shawn Squire, Nicholay Topin, and Kevin Winner. At Brown University: David Abel, Nakul Gopalan, and Lawson Wong. Additionally, I offer my thanks to the various others with whom I have been grateful to work, including MAPLE and IRAL lab mates, and members of the CS Matters project: Tristan Adams, Khalil Anderson, David Atlas, Tadewos Bellete, Matthew Bird, Michael Bishoff, Kayla Carrigan, Noah Carver, Justin Chavez, Kasra Darvish, Chris Dinh, Nathan Epstein, Megean Garvin, Ennis Golaszewski, Joe Greenawalt, Aniebiet Jacob, Caroline Kery, Nathaniel Lam, Josh Massey, Keith McNamara, Desiree Mercure, Michael Neary, Russell Nesbitt, Dianne O'Grady-Cunniff, Arielle-Cherie Paterson, Nisha Pillai, Jan Plane, Sid Pramod, Luke Richards, Emily Scheerer, Connor Shaffer, Gurpreet Singh, Jennifer Smith, Brian Spiegel, Jeremy Suon, Chantal Tan, Beatriz Tinoco, Puja Trivedi, and Taylor Webb. With each I owe a debt of gratitude, either as equal collaborators, as a mentee learning from your experience, or as a mentor passing along what wisdom I have gleaned along the way. It has been my privilege to work alongside you, and to receive your support.

To my father, mother, and brother: thank you for your endless, boundless love.

Table of Contents

List of Tables

List of Figures

At first sight experience seems to bury us under a flood of external objects, pressing upon us with a sharp and importunate reality, calling us out of ourselves in a thousand forms of action. But when reflexion begins to act upon those objects they are dissipated under its influence; the cohesive force seems suspended like a trick of magic; each object is loosed into a group of impressions–colour, odour, texture–in the mind of the observer. And if we continue to dwell in thought on this world, not of objects in the solidity with which language invests them, but of impressions unstable, flickering, inconsistent, which burn and are extinguished with our consciousness of them, it contracts still further; the whole scope of observation is dwarfed to the narrow chamber of the individual mind. Experience, already reduced to a swarm of impressions, is ringed round for each one of us by that thick wall of personality through which no real voice has ever pierced on its way to us, or from us to that which we can only conjecture to be without. Every one of those impressions is the impression of the individual in his isolation, each mind keeping as a solitary prisoner its own dream of a world. Analysis goes a step farther still, and assures us that those impressions of the individual mind to which, for each one of us, experience dwindles down, are in perpetual flight; that each of them is limited by time, and that as time is infinitely divisible, each of them is infinitely divisible also; all that is actual in it being a single moment, gone while we try to apprehend it, of which it may ever be more truly said that it has ceased to be than that it is. To such a tremulous wisp constantly re-forming itself on the stream, to a single sharp impression, with a sense in it, a relic more or less fleeting, of such moments gone by, what is real in our life fines itself down. It is with this movement, with the passage and dissolution of impressions, images, sensations, that analysis leaves off–that continual vanishing away, that strange, perpetual weaving and unweaving of ourselves.

–Walter Pater, *The Renaissance*

Chapter 1: Introduction

In daily life, we encounter novelty of all kinds: new sensations, unfamiliar words, and unusual objects. Most commonly, we identify these anomalies as peculiar combinations of recognizable perceptions; we do our best to interpret and react to them. Without our innate capacity for adapting in the face of such uncertainty, we would quickly become acquainted with catastrophic failure. Indeed, human adaptability could be regarded as crucial to intelligence. Likewise, to make intelligent decisions about the future depends directly upon our ability to generalize about our actions and their effects on the world around us. Even when thrown into the unknown, we seek out additional information, make predictions, form plans, and update our beliefs about the world.

How might a machine, directed by its programming, achieve these behaviors? Since its inception, the science of artificial intelligence has been guided by Alan Turing's perspective on this topic. In particular, he argued that debating the question "can a machine think?" is less significant and worthwhile than exploring a machine's potential to emulate a human reaction, namely, to adapt to circumstances as we do [162].

The field of machine learning formalizes these notions of adaptability under the broader topic of generalization, in which a machine uses data, prior knowledge or experience, to address new input. Learning, tabula rasa, inherently requires some form of

adaptation by which the internal state of a machine is changed. Generalizing from what has already been learned to handle new situations continually is the greater challenge under consideration. This type of inference may be as straightforward as interpolating a new value from a learned function, or as complex as placing a stone in the game of Go that causes, uncharacteristically, the human opponent to leave the room.[1]

I aim to tackle this latter case, the problem of *decision making*, when a machine applies what it has learned to produce meaningful effects in the world and repeatedly adapt to any feedback it observes. Specifically, in this and subsequent chapters, I make the following contention: **achieving more adaptable decision-making machines begins by representing and reifying abstractions of both perception and behavior**. The resulting investigation of abstract decision making in this thesis proceeds by analyzing techniques, as introduced in the following sections, for agents to reason about anomalies through concept formation and generalize their behavior through temporal abstraction.

1.1 Abstract Decision Making

A central aim of artificial intelligence is to create programs capable of rational action. Typically, these programs are restricted to a given setting, and are assigned an objective to optimize, such as minimizing some measurement of cost over time or attaining the greatest expected utility. Machines programmed for these ends are commonly referred to as intelligent *agents* [133].

Agents, like humans, require the ability to adapt to changing environment, both to

[1] This event is referred to as "Move 37" of the second game played by AlphaGo, a machine learning algorithm, against the world champion [60, 111].

learn and to operate robustly. This fact is particularly true for situated agents such as robots that function in the open, with real, physical elements. Imagine robots capable of adaptability, correctly reacting to an anomalous obstruction or deciphering novel commands from a human collaborator. Such a goal applies equally for software agents; even in systems with well defined bounds and parameters, it would be a reasonable desire to transfer those agents to new circumstances, evaluating unseen data under nonstationary conditions.

How might problems be framed to assess the intelligence of such agents? An assumption underlying this question is that an agent must make a decision: given some input data representing an object, observation, or situation, it generates some output as a number, label, grouping, delineation, or action. A human designer generally presents these tasks as either supervised, unsupervised, and reinforcement learning (RL). This third paradigm, RL, uniquely considers the scenario in which an agent's decisions affect what it next observes. Thus, in assessing the ability of agents to adapt and generalize intelligently, I approach solutions from the perspective of RL.

A key problem for RL agents is that they are most often engineered and trained for a specific task or setting alone. Generalization or adaptability to new situations is prohibitively costly or not possible given the agent's programming or internal representation of knowledge. Many possible and overlapping avenues of solutions are under exploration: function approximation [147, 172] (including deep RL, in which artificial neural networks serve as approximators [9]), task-centric transfer learning [28, 154], learning-to-learning and meta-RL [41, 138]. Each of these topics focuses on a particular angle, placing certain challenges and possible solutions at the fore. Similarly, in grappling with this goal of

generalization, I investigate *abstraction*, the creation of representations suitable for more generalized decision making,

1.1.1 Two Types of Abstraction

I promulgate a two-pronged approach for abstract decision making. An agent perceives its current state and executes its decisions; abstractions may be made from realizations of both the former (states) and the latter (actions). One might use the term *concept* to describe each of a state abstraction and an action abstraction, and *concept formation* as the process creating them [1]. Reasoning fully abstractly (based on input observations and output conduct) requires forming state concepts and action concepts.

Employing the term "concept" courts danger in its vague and widespread use across fields such as linguistics, psychology, and cognitive science. Concepts may be the fundamental constituents of thought, possibly innate, inextricably linked with language and mentally grounded to symbols like words [43,44]. Alternatively, they may be abilities that describe what the mind can do rather than what the mind represents [77]. Or, perhaps they are probabilistic prototypes from which we create categories [106]. Even bringing our focus upon RL literature, there is no consensus usage: concepts may be behaviors [62], programs [88], symbols [46,47], spatial relations [114], clusters of states [52], or clusters of points in state-action space [115]. From this diversity of application, I distill a more precise meaning for the remainder of this thesis.

State concepts clearly must capture some generalization about the agent's world. In contrast, action concepts express repeated patterns of behavior over which an agent

might reason. As such, they might more readily be called *habits*, as actions grouped into units that perform desired procedures. Following from the history of philosophy, the notions of concept and object are connected logically; Gottlob Frege's definition of concept is a mapping from an object to a truth statement about the properties of that object ("X is greater than 2," "X is happy," "X is a planet") [45, 174]. I adhere to this phraseology and apply the term "concepts" to mean abstractions based on world states and an agent's observations. Uniting these ideas together, I suggest they be considered broadly as concepts (abstractions of the perceptions) and habits (abstractions of behavior). In later discussions, I refer to the instantiations of concepts and habits respectively as *formal concepts* and *subtasks*, names more commonly used in literature.

1.2 Anomaly Reasoning through Concept Formation

A key problem in any decision-making situation is the question: how should an agent react to an anomaly? By *anomaly*, I refer to any novel or unexpected entity in the environment, with respect to an agent's purpose, experience, and typical milieu. Therefore, for practical purposes, anomalies broadly encompass any features, objects, or situations before which an agent has never seen.

The ability to reason about anomalies would constitute apprehending and suitably incorporating them into decision making. Such methods would necessarily improve the speed and quality of learning, minimize failure, and ensure versatility in changing, uncertain settings. This process of relating unknown phenomena to the known is closely related to the goals of knowledge transfer and lifelong learning, where the same agent

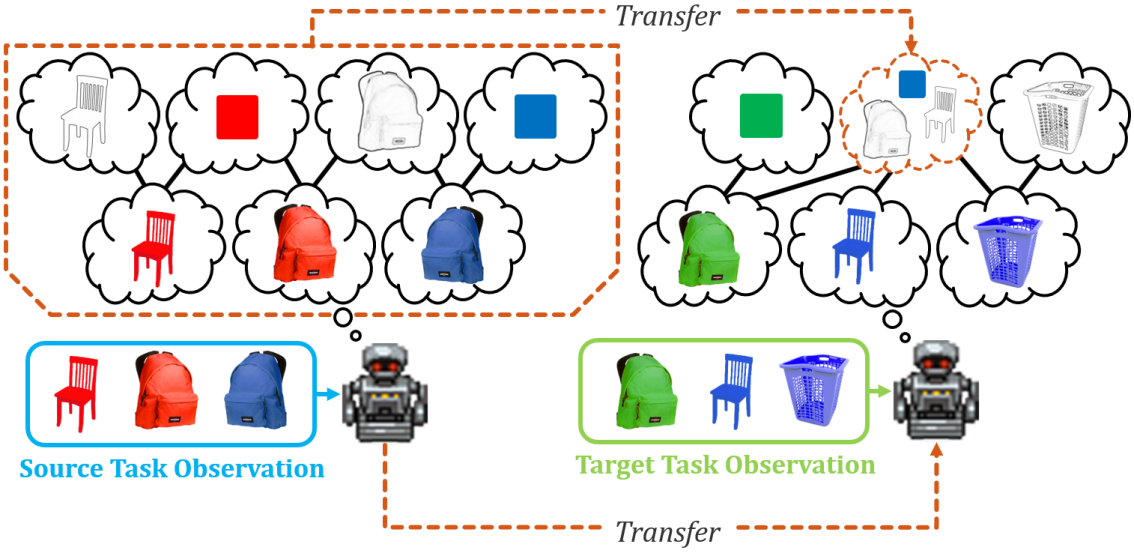


Figure 1.1: A schematic cartoon of concept formation. An agent forms abstract concepts from an observation in a source task, and generalizes that knowledge to anomalies observed in a target task. In this example, the agent learns about shapes and colors. Concepts are hierarchical: the sub-concept of "red chair" has the super-concepts of "red" and "chair." Likewise, that sub-concept is more conceptually similar to "red backpack" than "blue backpack." The agent learns behaviors associated with these concepts, and *transfers* this knowledge (denoted by the dashed orange lines) to a new task. It then interprets the new objects it encounters by relating unfamiliar concepts abstractly to what it has seen before.

reuses its experience across multiple tasks among various domains. I propose that sufficient anomaly reasoning requires a framework of three processes in sequence: identification, interpretation, and adaptation to anomalies. Each of these stages requires some form of explicit representation to consider new, anomalous observations in the light of prior experience.

To achieve this end, I contribute a framework and algorithms for agents to recognize and react to anomalies explicitly through *concept formation* [42]. The process of conceptualization, or forming concepts based on objects in the world, produces the state abstraction necessary for generalization as discussed in Section 1.1.1. To achieve anomaly reasoning, concept formation reshapes the perceptions into representations of an extensi-

ble, hierarchical knowledge base (of concepts), such that new perceptions are subsumed into this space. In particular, I leverage the theory of formal concept analysis (FCA) to generate and represent concepts extracted from arbitrary data [124].

A formal concept in FCA encapsulates a relevant cluster of features, relating objects to their properties, such that they possess membership in successively abstract sets. In this way, FCA theory parallels Frege's logical formulation of concepts and objects [45]. Each formal concept, then, is an emblem of an object subset paired with a subset of their shared properties. For decision making, these concepts serve as the internal abstractions of experienced phenomena, grouping features that agents may recognize, retain, and recombine to recontextualize anomalies into the space of what is known.

At a high level, I outline how agents may create concepts to store knowledge and more gracefully adapt to anomalies. By forming concepts, an agent essentially maintains an ever-growing ontology that connects decisions to meaningful properties of its world at various levels of abstraction. To perform anomaly reasoning, an agent perceives its surroundings, extracts conceptual features, and learns behaviors to thereby interpret any anomalies, for instance, by finding their closest-known analogue in conceptual space. Ultimately, all anomalies are progressively incorporated into an agent's understanding, mapped into this learned structure, interpreted, and assimilated appropriately. Concept formation, thus, offers an active bottom-up construction of a world model in which an agent retains learned objects, attributes, and the relations among them. Together, anomaly reasoning and concept formation unite to help agents attain knowledge transfer and lifelong learning while operating under uncertainty.

This approach combines machine learning and formal structures of knowledge, si-

multaneously making agents' decisions more explicable and granting the ability to transfer learned behaviors to tasks across environments where novelty and exogeneity – that is, anomalies – are possible. In this dissertation, I introduce a theoretical framework for anomaly reasoning through concept formation, and investigate how the symbolic and logical structures of FCA offer advantages in application to the statistical learning tasks of supervised classification, multi-armed bandits problems, and reinforcement learning.

1.3 Generalizing Behavior through Temporal Abstraction

Human knowledge is not limited just to existing objects and concepts derived from them; we also reason abstractly over our interactions with the environment, continually forming long-term plans and aiming to achieve distant, abstract goals. Enabling generalization over actions, hence, complements an intelligent agent's abstract understanding of the world via concepts. By learning how the structure and order of our actions affect the world around us, we make decisions by applying the principle of induction, reasoning about future causes and effects based on previous experiences.

Habit formation facilitates this process. A habit, the realization of action sequences into a whole object unto itself, alleviates the cognitive load of shuffling through the plethora of possible paths one might pursue. Habit, as a force solidifying convention and routine, is much maligned by such a range of thinkers as Francis Bacon, Ralph Waldo Emerson, Walter Pater, and Samuel Beckett, though it gains support from one key figure: William James [163]. As a progenitor of the science of psychology, James articulates his notion of habit, specifically that "habit simplifies the movements required to achieve

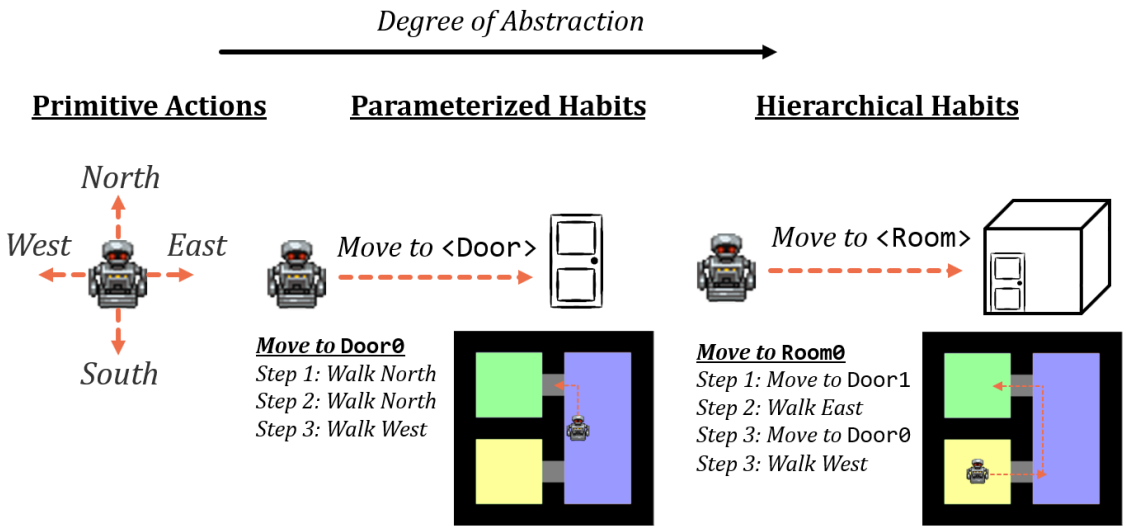


Figure 1.2: A schematic cartoon of habit formation. An agent forms increasingly abstract habits by generalizing its repeated behaviors. In this example, the agent plans navigation in cardinal directions, traveling among rooms separated by doors. Habits are parameterized: instead of going to a specific door, agents could learn the *subtask* of moving to any door, habitually. Example grounded rollouts are shown in visualizations of the Cleanup domain (see Chapter 5). Ultimately, agents may compose habits to construct hierarchies of more abstract ones, such as using the habit of going between doors to reach new rooms.

a given result" and "habit diminishes the conscious attention by which our actions are performed," casting these as crucial benefits [67]. Excessive deference to habit certainly breeds unthinking automatism in people. However, when enlisted for efficient over injurious ends, rather than undermining intelligence habit offers a basis upon which one finds the consistency and security to attempt more bold, creative, and exploratory endeavors. Precisely in this sense, I argue that a useful habit provides the building block by which a human or machine agent may bound from the local to the distant in time and space.

In the realm of artificial intelligence, this process of generalizing behavior for decision making is most frequently considered from the vantage of temporal abstraction [150]. For people, temporal abstraction means envisioning our progress over varying time scales. As a human agent operating in the world, one does not consider only the most low-level

actions possible, but also high-level patterns or sequences of actions. For example, in traveling from city to city, driving a car requires many minute, fine-grained motions. The planning and communication of an itinerary, however, invariably centers around landmarks such as route names, approximated distances, and expected times. A road trip across the country might be discussed at an even higher remove, such that our understanding of navigation occurs at successively more abstract periods of time.

Paired with temporal abstraction is the notion of a hierarchy of actions. At any given moment, the individual proximate actions one takes are usually in accord with some overarching purpose. Upon learning repeated patterns of actions as habits that accomplish a goal, it may be more formally stated that one further reasons about them as *subtasks* [64].[2] Composing subtasks yields a *task hierarchy*: a subtask comprised of subtasks is inherently temporally abstract, but may be reasoned about atomically.

Such abstract tasks are naturally complemented by state abstraction, focusing on objects in the world that are most important while ignoring those that are irrelevant. When tasked with driving a car, for example, one cares about the local space inside the car, such as one's posture in the seat; in thinking about how to reach one highway from another, however, such details are not factored into one's plans. It is precisely this combined, mutual benefit of state and temporal abstraction that I investigate in this dissertation, generalizing in the context of novelty. In particular, my approach formulates habits as subtasks in a representation conducive to being learned; I develop and assess algorithms for learning them simultaneously with various levels of state abstraction, using them to

[2] In this thesis, I defer to the common term for habits, "subtasks," though I suggest this more colloquial name aptly suits its usage in decision making.

plan abstractly over their expected duration, and grounding them to anomalous tasks.

1.4 Dissertation Outline

I now outline the structure of this dissertation, including a recapitulation of my problem statement and summary of contributions as they appear throughout the contents of this text.

1.4.1 Problem Statement

I aim to make and analyze techniques for improving the ability of artificially intelligent agents to generalize. The goal of generalization addresses a core problem of such agents, namely, that they may be trained solve a specific, narrow task, yet struggle to adapt to new challenges and preserve what they have learned. Currently, practitioners delineate subtopics of generalization like transfer and meta-learning, while applying different forms of function approximation to achieve generalization; I introduce these topics broadly in the next two chapters, and in later chapters I address the limitations of related work in context with my methods. Unique to my approach, I unite statistical machine learning with formal structures of knowledge so that agents explicitly create concepts and habits, which are reified abstractions of states and actions, respectively. By learning concepts, agents can subsume novel entities into their knowledge, achieving anomaly reasoning (correctly interpreting and reacting to something new). By learning habits, agents adapt better to new situations, more efficiently model the effect of their actions upon the world, and more effectively plan into the distant future. The ability to generalize is central to

human intelligence; improving it for agents makes them more robust and successful when facing uncertainty. Ultimately, I advocate a vision for these to threads to be united, and in the final chapter I lay out the agenda for future work to explore a more deeply entwined interaction of concepts and habits.

1.4.2 Trajectory & Summary of Contributions

I draw upon disparate areas of study, primarily reinforcement learning, probabilistic planning, function approximation, and concept formation. Consequently, I begin in Chapter 2 with an overview of the background necessary to understand these four topics. I specifically highlight the theories of Markov decision processes, multi-armed bandits, and formal concept analysis, as these form the basis of each future chapter. Chapter 3 surveys a broad range of related approaches that inspired and help contextualize the remainder of the thesis. Each of the subjects under discussion affect aspects of generalization and adaptability, especially the use of abstract, symbolic, and structured knowledge. I provide an overview of methods for state abstraction, transfer and lifelong learning, memory, case-based and analogical reasoning, anomaly detection, and concept-based learning.

Chapter 4 proposes a theoretical framework for reasoning about anomalies. The articulation of this problem is novel, and emphasizes the essential procedures necessary to identify, interpret, and adapt to anomalies. The discussion revolves around FCA, especially the application of formal concepts as vehicles for finding abstract relations among observed entities. Formal concepts possess advantageous properties for doing so: they are innately hierarchical, make minimal assumptions about the structure of input data, and

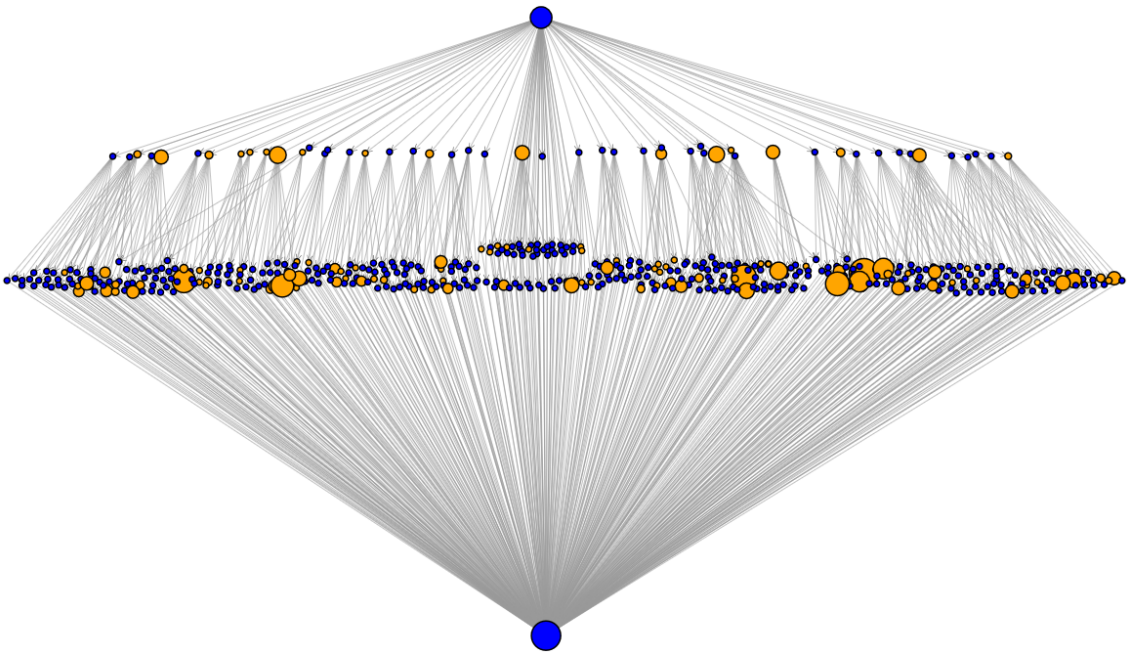


Figure 1.3: An example concept meta-graph. In Chapter 5, I explain how I define this hierarchical structure. It expresses the relationship between abstract, formal concepts (nodes). The meta-graph shown here is assembled from a single trial of 300 episodes of the Synthetic CB problem (see Section 5.2.5.2).

permit an automatic extension to any newly observed features. I conclude by showing how a basic anomaly interpretation problem can be modeled as a classification task, and demonstrate the viability of a formal concept approach to mapping unknown to known objects.

Chapter 5 introduces concept-aware decision making, where state abstraction is handled through concept formation, promoting the transfer of knowledge so agents may better adapt to uncertainty. First, I build upon the theory of anomaly reasoning from Chapter 4, extending it to an object-oriented Markov decision process setting, suitable for both RL and planning. I call this contribution *concept-aware feature extraction* (CAFE) [170]. After discussing its properties as the basis for a form of function approximation, I first examine CAFE in a multi-armed bandits scenario. Dubbing the generic method *concep-*

tual bandits with concepts (CBC), I explore an implementation of it with hybrid linear models, based on LinUCB [90]. These results include a static synthetic data problem, for which concepts would be ill-suited, and more realistic tasks of increasing difficulty that highlight the benefits of CAFE when dealing with anomalies. In these more complex domains, anomalous objects and contexts are ubiquitous; CBC agents evince an improved ability to adapt and generalize. To understand the learned structure of concepts more clearly, I also develop a novel visualization technique for viewing the latent *concept metagraph*. Proceeding into RL, I discuss CAFE as applied to temporal difference algorithms: *concept-aware reinforcement learning* (CARL). Knowledge transfer becomes even more challenging in this paradigm, as the decisions that are made based on concepts also affect future performance. As with CBC, I find CARL algorithms more gracefully handle the variety and uncertainty in new environments. These contributions are articulated fully in this thesis; I intend each to constitute the basis of future publications.

In the final set of chapters, hierarchical decision making is extended with new methods that help agents learn to generalize abstract actions played out over time. Chapter 6 outlines the foundations of habits as temporal abstractions. These hypostatizations are represented by a hierarchical organization of subtasks. Starting from an overview on existing methods for bottom-up learning and transferring subtasks, I articulate my contributions to the topic, from work with colleagues: *the expected-length model of options* (ELM) [4], and *portable option discovery* (POD) [159]. Chapter 7 covers additional related work on temporal abstraction from the top-down view. There, I focus on the background behind joint work on *abstract Markov decision processes* (AMDPs) [55], in which agents formulate and reuse long-term plans at multiple levels of abstraction in a task hier-

archy. Chapter 8 unites the ideas of state abstraction, bottom-up learning, and top-down planning into a singular approach: *planning with abstract, learned models* (PALM). With PALM, agents discover, from experience, the abstract representations and structures that permit transfer and generalization for subtasks over time. This work is in submission to be presented as a conference paper this year, and is based upon preliminary model-based extensions of AMDPs [171].

Finally, Chapter 9 concludes the dissertation and discusses future work that further builds upon state and action abstractions, synthesizing the presented approach of concepts and habits towards more generalized, abstract decision making. Combined, my contributed efforts endeavor to help make agents better generalize under uncertainty. By acquiring knowledge representations suited for transferring their experience to new tasks, these agents may be said to be more adaptable and intelligent.

Chapter 2: Background

This dissertation discusses an arc of research covering approaches to reasoning about anomalies, concept formation, and abstractions over state space and time. The foundation is established in this chapter, which introduces agent-based methods of decision making (reinforcement learning and planning) and an information-theoretic approach of representing an ontology of objects and their attributes (formal concept analysis).

2.1 Markov Decision Processes

Reinforcement learning (RL) and probabilistic planning are related paradigms for solving decision-making problems where an intelligent agent makes an observation, interacts with the environment, and receives feedback in the form of a reward or punishment [128, 148, 149]. Stochastic decision-making problems are commonly described by Markov decision processes (MDPs), in which observations are defined as *states*.

Definition 2.1 (Markov Decision Process). *The finite Markov decision process (MDP) is defined as a five-tuple:*

$$M := \langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle,$$

containing a set of states $\mathcal{S}$, a set of actions $\mathcal{A}$, a transition probability distribution function T, a reward function R, and a constant scalar discount factor γ [20].

States are assumed to be fully observable, encompassing the totality of the environment. Therefore, states do not possess any inherent uncertainty, hidden information, or latent parameters; states with such properties are encompassed by the larger class of *partially-observable* MDPs (POMDPs). Action sets may be restricted, with certain actions not permitted or available in certain states. The transition probability T is a function of the form $T(s, a, s') = \Pr(s'|s, a)$ of entering the next *successor* state $s' \in \mathcal{S}$ upon executing action $a \in \mathcal{A}$ from *source* state $s \in \mathcal{S}$. A simple transition is considered to occur in one *time-step*. The function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ expresses the feedback received when following a transition (s, a, s'), such that real number $r = R(s, a, s')$ is the immediate reward of the transition at that time-step. The parameter $\gamma \in [0, 1]$ is known as the discount factor, representing an agent's preference of immediate reward relative to future rewards discounted geometrically (as in Equation 2.1). A γ of 0 has an agent only consider the current reward when updating, where values of γ approaching 1 increase the agent's emphasis on rewards in future time-steps.

A crucial mathematical assumption of MDPs is the *Markov property*, that the transition probability of the successor state s' and expected reward r are conditionally dependent only on the current state s and action a. Thus, in general, decision-making algorithms assume only access to the current state. Many techniques do exist for augmenting an agent with memory, such as by storing a history of sampled transitions from which to perform batch updating or experience replay, though these only affect learning (the Markov property is assumed to hold, and the underlying MDP mechanics remain unchanged).

MDPs as Tasks. An MDP poses a formal representation of a scenario, or *task*, for the agent to solve. Without sacrificing an MDP's generality, the specification may include a set of terminal states $\mathcal{E} \subset \mathcal{S}$ in which an agent's episodic learning would terminate. $\mathcal{E}$ could capture, for instance, cases of success and failure for the task at hand. In later discussions of MDPs, the terms *domain* and *task universe* interchangeably describe a family of related tasks. A domain, then, may be viewed as a distribution of possible tasks from which a specific task MDP is drawn.

RL vs. Planning. RL differs from planning in the task knowledge that an agent is assumed to have. In particular, planning tasks provide an explicit model of the transition dynamics and reward (such as the true T and R of the MDP). RL agents, however, lack direct access to those functions.

Solving MDPs. An agent typically attempts to solve an MDP by finding a *policy*, π, some function that specifies how an agent should behave in a given state. The solution to any MDP is the optimal policy, π^*, that maximizes the expected discounted future rewards. A deterministic policy is a mapping of states to action, $\pi : S \rightarrow A$. More generally, a stochastic policy $\pi : S \times A \rightarrow [0, 1]$ expresses the probability $\pi(s, a)$ that the agent should take a while in s. Algorithms find π^* by computing either V, the *value function* (expected value of discounted rewards, given starting state s while following π), or Q, the *action-value function* (expected value of discounted rewards, while taking a in s and thereafter following π).

Definition 2.2 (Value Function). *For a given MDP, the value function is defined under a policy π as:*

$$V^{\pi}(s) := \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s\right], \tag{2.1}$$

The value function captures the notion of a state's utility, based on the average discounted future rewards that would be received by taking the actions specified by π.

Definition 2.3 (Action-Value Function). *For a given MDP, the action-value function is defined under a policy π as:*

$$Q^{\pi}(s,a) := \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \,\middle|\, s,a\right]. \tag{2.2}$$

Finding the optimal value function V^* or action-value function Q^* induces an optimal policy. To acquire π^* from Q^*, an agent may simply take the action with the max Q-value at any state. From V^*, the agent need only perform a greedy one-step search over subsequent states and take the action corresponding to the max value found. Both V^* and Q^* adhere to Bellman optimality and may be updated via Bellman backups according to the rule:

$$V^*(s) = \max_{a\in\mathcal{A}} \sum_{s'} T(s,a,s')\big[R(s,a,s') + \gamma V^*(s')\big], \tag{2.3}$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')\big[R(s,a,s') + \gamma \max_{a'\in\mathcal{A}} Q^*(s',a')\big]. \tag{2.4}$$

Value Iteration (VI). VI is one classic planning algorithm that computes the optimal value function by applying the Bellman backup of Equation 2.3 to all states repeatedly until the max change in value falls below some error threshold. VI converges to the

optimal value function (assuming the MDP is finite and the threshold is low enough), but is slow given that every iteration requires $|\mathcal{S}|$ backups [149].

Q-learning (QL). QL [167] is a standard RL algorithm that computes and converges to Q^* from an arbitrary initialization of $Q(s, a)$ and a given learning rate α. It is an "off-policy" approach in the sense that it can yield π^* while following another π. Most commonly, QL follows a policy employing the ϵ-greedy strategy, where an agent takes a random action with some small ϵ probability and otherwise selects the action with the maximum Q-value. For some number of episodes (or until convergence), a QL agent selects an action according to its policy, obtains reward r and the next state s', and then computes the update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a' \in \mathcal{A}} Q(s', a') - Q(s, a)]. \tag{2.5}$$

Transferability. The optimal policy, value, and action-value functions are specific to the dynamics of the given MDP, with a recursively dependence on future states, transition probabilities, and reward schedules. Thus, they are not inherently transferable to other MDPs, making generalization to new tasks challenging. Throughout this dissertation I aim to tackle this problem, and discuss the creation of abstractions and knowledge representations that facilitate transfer of plans and learned behaviors. In the next chapter, I discuss previous and related work on dealing with transfer in RL.

2.1.1 Object-Oriented Markov Decision Processes

The object-oriented Markov decision process (OO-MDP) is a factored state space variant of the MDP where states are composed of objects whose behavior is defined by the instantiated values of their attributes [36]. In general, MDPs with a factored state space represent states by vectors of *factors*; OO-MDPs encode basic domain knowledge to enforce greater structure over the allowed values of factors.

Definition 2.4 (Object-Oriented Markov Decision Process). *A finite object-oriented Markov decision process (OO-MDP) is defined as:*

$$M := \langle \mathcal{S} \leftarrow \{\mathcal{C}, \mathcal{O}, \mathcal{B}, \mathcal{D}, \mathcal{M}\}, \mathcal{A}, T, R, \gamma \rangle,$$

containing the same components of an MDP (Definition 2.1), except that the set of states implicit, generated from the relations among a set of classes $\mathcal{C}$, a set of objects $\mathcal{O}$, a set of attributes $\mathcal{B}$, a set of attribute domains $\mathcal{D}$, and a set of attribute-values $\mathcal{M}$.

A given OO-MDP has n objects, $\mathcal{O} = \{o_1, \ldots, o_n\}$. Each object is a member of some class, $\forall\ o \in \mathcal{O},\ Mem(o) = C \in \mathcal{C}$. Every C defines the template of attributes that its member objects possess. Thus, C is the set of k attributes for that class, $Att(C) = \{C.b_1, \ldots, C.b_k\} \subseteq \mathcal{B}$. An attribute has its own domain bounding the possible attribute-values that may be assigned to it, $Dom(C.b) = \{m \ldots\} \subseteq \mathcal{M}$. Thus, the set of attribute-values $\mathcal{M}$ simply expresses the space of all possible factor values. At a specific time-step, an object has a realized *object-state*, the list of all current attribute-values it possesses (one for each attribute of its class). An OO-MDP state is vector of factors, consisting of the union of every object's object-state, $s = \cup_{i=1}^{n} \text{OBJECT-STATE}(o_i)$.

The main benefits of the OO-MDP formalization are that it allows more general,

expressive, and easily extensible definitions for decision-making problems. OO-MDPs may specify class-level relations, or a set of predicates $p \in \mathcal{P} : s \in \mathcal{S} \rightarrow \{0, 1\}$ that provide higher-level information such as relations among objects. In this sense, the OO-MDP grants a natural way of describing environments as a collection of objects, their properties, and their relations. Additionally, objects or attributes can easily be added or removed, making the the OO-MDP formulation ideal for state abstraction.

2.1.2 Function Approximation

RL has been employed notably in learning to play games such as Backgammon [157], Atari video games [113], and Go [140], achieving human and super-human levels of performance on them. In each of those cases, the technique of *function approximation* was key to addressing the issue of generalizing experience over the immense state space of the games. When the number of possible states is on an order far exceeding what can be recorded feasibly, a standard approach is to approximate the value or action-value function. The linear form of value function approximation (VFA) follows:

$$\hat{V}_\theta(s) = \theta^\mathsf{T}\phi(s) \qquad \text{and} \qquad \hat{Q}_\theta(s, a) = \theta^\mathsf{T}\phi(s, a). \tag{2.6}$$

In VFA, the core mechanisms are a vector of weight parameters θ and a vector of basis functions $\phi(s)$ or $\phi(s, a)$ [49]. The basis functions serve as the set of *features* representing a given state; an overcomplete basis, where the set of vectors would still form a basis if one was removed, can offer more expressive function approximation. Rather than computing and storing the exact value or action-value for states, algorithms maintain this

weight vector. Weights are learned over time to approximate the relative contribution of a corresponding feature to the value or action-value function. Bootstrapping is a principle that has algorithms use their immediate predictions as the target, then sample and correct their estimates by adjusting them in the direction of the sampled value (thereby learning and improving while operating). In the bootstrapping technique of most standard temporal difference algorithms such as VI and QL, the agent observes a transition tuple at step t, (s_t, a_t, r_t, s_{t+1}). It learns the approximation by computing gradient descent (to minimize the cost) via parameter-wise Bellman updates:

$$\theta_{t+1} = \theta_t + \alpha_t\big(r_t + \gamma\hat{V}_{\theta_t}(s_{t+1}) - \hat{V}_{\theta_t}(s_t)\big)\big(\nabla\hat{V}_{\theta_t}(s_t)\big), \tag{2.7}$$

or

$$\theta_{t+1} = \theta_t + \alpha_t\big(r_t + \gamma\max_{a\in\mathcal{A}}\hat{Q}_{\theta_t}(s_{t+1}, a) - \hat{Q}_{\theta_t}(s_t, a_t)\big)\big(\nabla\hat{Q}_{\theta_t}(s_t, a_t)\big), \tag{2.8}$$

for iteration t, where α_t is the learning rate, and r_t is the reward from the transition. These RL methods are in essence performing supervised learning where transition tuples are data and the value or action-value are the target signal [48]. Temporal difference algorithms such as Q-learning and SARSA with function approximation have been proven to converge when certain conditions are met [109, 161].

Common forms of featurization for linear value function approximation include tile-coding (also known as cerebellar model articulator controller, CMAC) [5], polynomial basis, or Fourier basis functions [84]. For example, given state vector $\bar{x} = [x_0, x_1]$, a degree-2 polynomial basis function expansion would be $\phi(\bar{x}) = [1, x_0, x_1, x_0x_1, x_0^2, x_1^2]$. Tile-coding approximates functions by creating discrete "tiles" over continuous variables,

often applying multiple, overlapping offset tilings. The features are then represented as a high-dimensional but sparse binary vector, with elements set to one to denote tiles containing the featurized variable. A downside to tile-coding is that the range of variables must be known beforehand, and it requires some engineering to select the appropriate degree of granularity and overlapping, although adaptive versions exist [169]. Alternate schemes for featurization include online construction of features. One such example is incremental feature-dependency discovery (iFDD), which gradually expands the space of features. In iFDD, new features are constructed as combinations of existing features. The temporal difference error of potential new features is assessed, and they are added to feature space once their value surpasses some given threshold [50]. The concept-aware feature extraction outlined in Section 5.1 is adaptive in a way similar to iFDD, but it is not incremental and does not assess utility of concepts before adding them to the feature space. Adaptive methods such as iFDD are more expressive, leading to useful feature discovery, faster convergence, and better policies especially in larger domains [51].

2.2 Multi-Armed Bandits

Closely related to Markov decision processes is the sequential decision-making problem commonly referred to as *multi-armed bandit* (MAB). This name refers to the colloquial epithet of slot-machines: one-armed bandits. MABs present a "slot machine" with n arms (hence, multi), such that any given state has n actions, one for pulling each arm, with reward determined probabilistically, conditionally dependent on the arm pulled. The goal, then, is to find the optimal policy that pulls the arm with the greatest return. Central

to the MAB problem is the core exploration-vs-exploitation challenge of RL. An agent for MABs invariably requires some degree of sampling of each arm, to acquire knowledge of the arms' reward distribution. Ultimately, however, an agent should exploit. In the simplest MAB tasks, this practice means committing to one action and only pulling that arm; more complex behavior relies on probabilistic policies conditional on the state or set of arms available. MABs can be thought of as a one-step MDP, with the crucial element being that successive states do not depend on the previous ones. In other words, there are no transition dynamics; states in MABs are selected randomly, arbitrarily, or adversarially depending on the type of problem. In the minimal version of MABs, states have no distinguishing features and may be ignored or considered identical, or they merely specify which arms of the action set are available (if not all of them). Myriad extensions to MAB exist, and I primarily focus on the case where states do differ, and the behavior of arms and rewards depend upon them (see Section 2.2.1).

I present my framing of the general MAB problem as $\langle \mathcal{S}, \mathcal{A}, R \rangle$, with a set of states, an action set (the arms), and a reward function. Generally, states are framed as originating from an adversary, either obliviously (selected at random) or by following some agenda [87]. In the case that $\mathcal{S} = \varnothing$, the reward function depends solely on the arms, $R : A \to \mathbb{R}$. With states where some arms may be available at different times, consider $R : S \times A \to \mathbb{R}$ such that this reward function R is undefined on any unavailable arm a, as restricted by s. In experiments, an agent at time t observes a state s_t (if one exists) and the set of available arms $\hat{\mathcal{A}}_t \subseteq \mathcal{A}$. It then selects an arm $a_t \in \hat{\mathcal{A}}_t$, and observes scalar reward $R(s_t, a_t) = r_t$ received from pulling a_t in state s_t. Altogether, multiple executions yield a *history* of episode tuples of the form (s_t, a_t, r_t). The notion of an

action-value from RL is reused here as $Q(s_t, a_t) = \mathbb{E}[r_t | s_t, a_t]$. Likewise, the optimal value is $V^*(s) = \max_{a \in \hat{A}_t} Q(s_t, a)$.

As with MDPs, performance can be measured by cumulative expected reward. However, a more common metric is to compute a form of loss referred to as *regret* [87]. At a high level, regret is the difference between how well an algorithm might have been expected to perform and how well it actually did perform. It is necessary to distinguish between the single-step realized regret of a policy at time t, $\bar{r}_t^\pi$, and the cumulative expected regret over a history of n episodes, $\bar{R}_n^\pi$. The immediate regret of policy π adheres, in general, to the form $R(s, \pi^*(s)) - R(s, \pi(s))$. The expected single-step regret is defined as $\bar{r}_t^\pi = \mathbb{E}_\pi[V^*(s) - Q(s, a)]$. Formally, this dissertation specifies cumulative total regret as:

$$\bar{R}_n^\pi = \mathbb{E}_\pi \left[\sum_{t=0}^{n-1} \Big(V^*(s_t) - Q(s_t, a_t) \Big) \right]. \tag{2.9}$$

Therefore, $\bar{R}_n^\pi$ captures how much the agent's executed policy falls short of what it would have seen had it followed an optimal policy. Under this definition, all regret is non-negative. In plotting an algorithm's regret, a linear trend indicates no learning has taken place. Ideally, the results would be realized in a logarithmic or similar sub-linear pattern for successful algorithms.

2.2.1 Contextual Bandits

The contextual bandit (CB) problem is a reframing of MAB to include a *context*, or explicit factored state, upon which the reward function depends. I define CBs in the same framing as MABs, with the specification that states do exist, and contexts replace states

in the previously defined equations. At time t, a context x_t is a vector of features ϕ observed from the underlying state s_t, such that $x_t = \phi(s_t)$. Thus, regret is computed as in Equation 2.9, just with x_t in place of s_t. Generalization hinges upon this function approximation and on modeling probability distributions in reference to the context's features (that is, the probability of observing reward given the arm and context). In particular, this dissertation examines the *stochastic linear bandits* variation of contextual bandits [87]. Following the same linear function approximation discussed in Section 2.1.2 to represent value, stochastic linear bandits approaches the probabilistic estimation of expected CB reward, or action-value, for action a_t as:

$$\hat{Q}(s_t, a_t) = \theta_{a_t}^{\mathsf{T}} x_t. \tag{2.10}$$

Hence, the goal of CB algorithms is to learn parameters for each arm, θ_a, such that the reward model for action a approaches the expected value of the true reward function given that action, $\lim_{t\to\infty} \hat{Q}(s_t, a) \approx \mathbb{E}[R(s, a)|a]$, thereby facilitating the selection of actions to minimize regret. In literature, CB algorithms have found many practical uses, especially in preference-based or personalized services such as determining web advertisement placement or news article recommendations [90, 176] and dynamic search [173].

A CB context contains features that characterize the different arms available. That is, the state s_t contains the arms and any additional "side" information, such that ϕ is the feature-map or state abstraction mapping those arms into the context, a vector of perceptions. In this sense, an arm is like a class of an OO-MDP, where specific arms in the context are like instances of objects in an OO-MDP state. Further, the features of the

arms serve exactly as attributes of OO-MDP objects, meaning their instantiated values act exactly as attribute-values.

One primary difference between the setup of CBs and traditional MDPs is that full observability is typically not assumed. For example, arms may possess latent parameters not specified in the context which nevertheless determine their respective reward distributions. Alternatively, an unobserved "label" may be attached to the arms, and the agents goal could be to select an arm of the appropriate label given the observed context. This formulation shows how one might consider CB problems as occupying a place between supervised learning and MDPs: a CB agent is making decisions, but doing so by a kind of classification of contexts given the arms available [72].

CB-learning algorithms differ from standard RL ones such as QL in that they inherently require generalization across contexts. That is, the dynamic programming methods for MDPs rely on the Bellman update, where information about the value of one state propagates to others (due to the recursive definitions of value and action-value functions). Since successive contexts do not depend on previous decisions, a CB algorithm cannot deploy this approach. Contexts can be thought of as a row in a data set, as in supervised learning, except that the signal observed corresponds only to the arm pulled. Crucially, the counterfactual reward, or "what would have been received pulling a different arm," is unknown in the MAB problem, making loss much harder to assess than the error in classification tasks. Therefore, CB algorithms must combine generalization across contexts with robust exploration techniques.

In deciding which arm to select under uncertainty, the standard solution is the meta-algorithm known as Thompson sampling (TS) [158]. In TS, the algorithm applying it

is assumed to possess some mechanism for approximating the posterior probability (of policies or a reward model, in CB), whether through a frequentist or Bayesian perspective. Succinctly, TS simply samples parameters from the posterior distributions (one corresponding to each arm), selects the best arm based on the expected reward given the context (with respect to the conditional probability of the parameters), and updates its posterior based on both the parameters used and the reward received. For this reason, TS is sometimes referred to as Posterior Sampling. TS is notable in having been developed in the early twentieth century, only to languish until this century, becoming the de facto exploration method for bandits over the past fifteen years [134].

Another common approach to exploration applies the principle of "optimism under uncertainty." This strategy treats unknown or unmodeled processes as optimal, thereby greatly encouraging the underlying algorithm to explore them [87]. The Upper Confidence Bound (UCB) meta-algorithm is an application of this principle for CB algorithms. Optimism guides action selection in UCB: an overestimation of the unknown expected reward for each arm is maintained, and the arm with the highest estimate is selected. For MABs generally, UCB updates a confidence interval c_{t,a_t} at each time-step such that $|Q(s_t, a_t) - \hat{Q}(s_t, a_t)| < c_{t,a_t}$ is likely. UCB recommends the next action by computing:

$$a_{t+1} = \underset{a \in \mathcal{A}_{t+1}}{\arg\max} (\hat{Q}(s_t, a) + c_{t,a}). \tag{2.11}$$

Under the assumptions of linear stochastic bandits, the algorithm employing UCB is referred to as LinUCB [87, 90]. In this case, UCB must compute a confidence region over parameter space for each arm's model. Given parameter vector θ_t with m features,

the confidence region is a subset of parameter space, $\mathcal{C}_t \subset \mathbb{R}^m$. The purpose of $\mathcal{C}_t$ is to bound the space of possible θ such that the optimal values, θ^*, are contained. As samples are collected by the CB algorithm, the confidence region shrinks and the probability that θ^* is within it increases. A least squares method is needed to determine the confidence intervals surrounding the estimated parameters. For least squares, LinUCB applies ridge regression given a batch data set (X_a, ρ_a) for each action a, where context matrix X_a is $n \times m$ (n contexts each with m features), and $\rho_a \in \mathbb{R}^n$ is a vector of rewards observed when arm a was pulled in the corresponding context of X_a. Following the standard form of ridge regression, the parameters of arm a are estimated as:

$$\theta_a = Y_a^{-1} X_a^{\mathsf{T}} \rho_a, \tag{2.12}$$

with matrix $Y_a = X_a^{\mathsf{T}} X_a + I_m$ using the $m \times m$ identity matrix I_m. It has been proven in RL literature that the ridge regression of Equation 2.12 probabilistically satisfies an inequality [166]:

$$|Q(s_t, a_t) - \hat{Q}(s_t, a_t)| = |\mathbb{E}[r_t | x_t, a] - \theta_a^{\mathsf{T}} x_t| \leq \alpha \sqrt{x_t^{\mathsf{T}} Y_a^{-1} x_t}, \tag{2.13}$$

to some desired degree of uncertainty, $\alpha = 1 + \sqrt{\ln(2/\delta)/2}$, with a tuneable probability of error $\delta > 0$ such that Equation 2.13 decreases in probability as δ increases [90]. The upper bound of this inequality expresses the confidence region precisely as α standard deviations, such that LinUCB selects the best arm as in Equation 2.11 with $c_{t,a} = \sqrt{x_t^{\mathsf{T}} Y_a^{-1} x_t}$. In summation, LinUCB models the expected reward of each arm given a context while

maintaining an upper confidence bound estimate of each arm's model, following the principle of optimism under uncertainty by selecting the arm that appears best with respect to the current context.

2.3 Formal Concept Analysis

Formal concept analysis (FCA) is a mathematical technique based on information theory and order theory that has been growing in popularity in recent years [123, 124]. FCA provides a means of extracting knowledge structures from data in the form of formal concepts (paired sets of objects and attributes). Taken together, the concepts extracted from a set of data yield a partial ordering called a concept lattice that captures a hierarchical relation of concepts from the most abstract to the most specific.

A *formal context* is a triple $(\mathcal{G}, \mathcal{M}, \mathrm{I})$, with $\mathcal{G}$ the set of objects, $\mathcal{M}$ the set of attributes, and an incidence relation $\mathrm{I} : \mathcal{G} \times \mathcal{M} \rightarrow \{0, 1\}$ such that $\mathrm{I}(g, m) = 1$ when object $g \in \mathcal{G}$ possesses attribute $m \in \mathcal{M}$, else $I(g, m) = 0$. A context can commonly be represented as a binary matrix where rows are the objects, columns are attributes, and each element indicates the presence or absence of an attribute in the respective object. Let us define the concept-formation operators $\uparrow$ and $\downarrow$ to provide a way of mapping from objects to attributes and vice versa. Given a set of objects $A \subseteq \mathcal{G}$ and a set of attributes

$B \subseteq \mathcal{M}$:

$$A^{\uparrow} = \{m \in \mathcal{M} \mid \forall a \in A, \mathrm{I}(a, b) = 1\} \tag{2.14}$$

and

$$B^{\downarrow} = \{g \in \mathcal{G} \mid \forall b \in B, \mathrm{I}(a, b) = 1\}. \tag{2.15}$$

For an object $g \in \mathcal{G}$, $\{g\}^{\uparrow}$ is the set of all attributes present in g; for an attribute $m \in \mathcal{M}$, $\{m\}^{\downarrow}$ is the set of all objects possessing m. Then, a *formal concept* is defined from $(\mathcal{G}, M, I)$ as a pair of sets (A, B) such that $A = B^{\downarrow}$ and $B = A^{\uparrow}$. That is, for a formal concept (A, B), A is the set of objects possessing all attributes in B, and likewise B is the set of attributes shared by all objects in A. A is called the *extent*, and B is called the *intent*. The number of objects in the extent is referred to as the *support* of the concept. Another way of viewing formal concepts is as biclusters that are maximally inclusive on both the object and attribute sets [164]. Multiple algorithms exist for mining formal concepts from a context, such as FastCloseByOne and In Close2 [7].

Obtaining the formal concepts from a context inherently yields a partial ordering called a *concept lattice*, $\mathfrak{B}(\mathcal{G}, \mathcal{M}, \mathrm{I})$ with partial ordering $\leq$ of subconcepts to superconcepts ($\langle \mathfrak{B}, \leq \rangle$ is a complete lattice). In particular, formal concepts are ordered from the unit element (the top $\top$, a paired set of all objects and any attributes found in all objects) to the zero element (the bottom $\bot$, all attributes and any objects that possess all attributes). Hence, $\bot \leq \top$, and when a concept $(A, B) \leq (C, D)$ one says (A, B) is a more specific subconcept of the more general superconcept (C, D). A lattice can be visualized graphically where each node is a formal concept and the arcs express the natural

sub- and super-concept relationships. Objects are inherited down from the unit concept; attributes are inherited upwards from the zero concept. That is, for any node in the lattice, the union of its parents' attribute sets is a subset of its own attribute set, and the union of its children's object sets is a subset of its own object set. The unit and zero concepts can be viewed as complementary: the top contains all objects, the bottom contains all attributes. Moving down a lattice leads to more specific, reified concepts, while moving up leads to more abstract ones.

One important property is that the mapping that results from composing the concept-formation operators, mapping $A \to A^{\uparrow\downarrow}$ or $B \to B^{\downarrow\uparrow}$, induces a closure operator. The closure operator can find the least extent to which an object g belongs, such that $(\{g\}^{\uparrow\downarrow}, \{g\}^{\uparrow})$ is the most specific concept containing g, which is called the *object concept*. Similarly, the *attribute concept* of attribute m is the most general concept containing m in its intent, $(\{m\}^{\downarrow}, \{m\}^{\downarrow\uparrow})$. Object concepts are useful for interpreting anomalies since their extent contains grounded objects, so they provide a starting point to begin considering more abstract groupings of objects and attributes.

As an example of FCA capturing natural hierarchies of objects, consider the formal context in Table 2.1, a data set of species and their attributes (the Hasse diagram of the concept lattice built from it is shown in Figure 2.1). The lattice includes many semantically meaningful groupings. For instance, dogs and cats group together as having four legs, and both group with whales as having lungs and vertebrae. Lobsters and ants are grouped as invertebrates, and all group into the unit concept (where no attributes differentiate them), whereas none would group in the zero concept (as no species possesses all possible characteristics).

name	has-legs	four-legged	has-lungs	has-vertebrae	invertebrate	terrestrial	aquatic
dog	1	1	1	1	0	1	0
cat	1	1	1	1	0	1	0
whale	0	0	1	1	0	0	1
lobster	1	0	0	0	1	0	1
ant	1	0	0	0	1	1	0

Table 2.1: An example formal context of animal species and attributes.

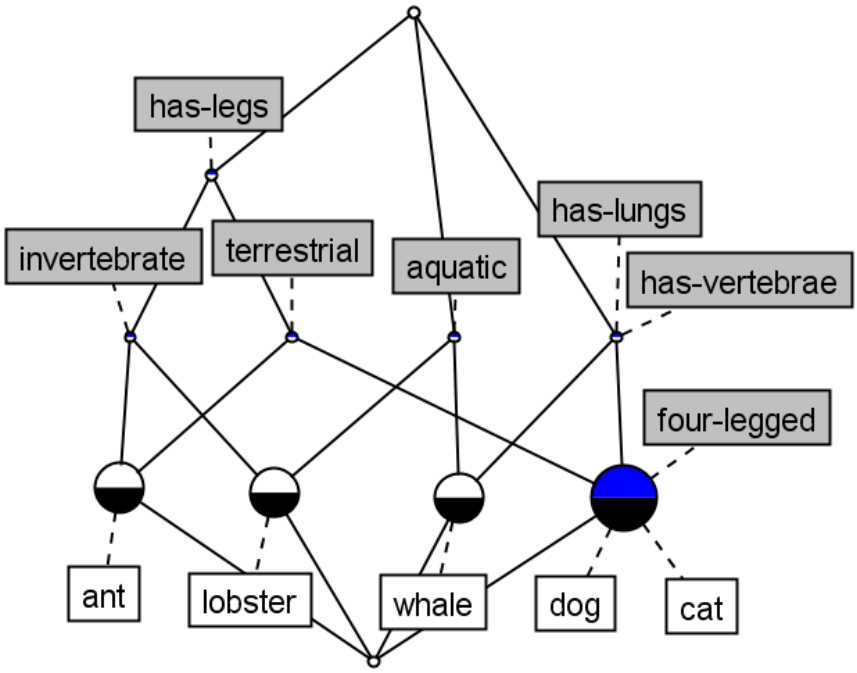


Figure 2.1: An example Hasse diagram of a concept lattice derived from Table 2.1. Nodes in the graph are formal concepts. Objects are inherited down from the top (root concept), and attributes are inherited in reverse from the bottom concept. Thus, attribute labels (in gray) are attached to the highest concept for which their respective attribute is a member, and object labels to the lowest. For example, the leftmost child of the top concept in the image is the concept of $\big(\{\text{ant, lobster, dog, cat}\}, \{\text{has-legs}\}\big)$, which one might semantically define as "legged animals." The leftmost child of that concept in the image is $\big(\{\text{ant, lobster}\}, \{\text{has-legs, invertebrate}\}\big)$.

FCA is most commonly used for mining static data sets such as text corpora for semantic relations. This dissertation presents a first approach to employ FCA interactively in an agent-based decision-making context. The motivation for conveying agent knowledge through formal concepts is that they are descriptive yet small and hierarchical, arising simply from a data set itself. Moreover, a lattice provides a type of natural unsupervised clustering of objects in an agent's world, forming a kind of ontology, where its concepts are informative groupings of perceptions and components of the world that can be used to reason and learn.

Chapter 3: Related Work

This dissertation considers new methods for abstract decision making, with a focus on adaptability, especially the problems of generalizing behavior over time and reasoning about anomalies through concept formation. I aim for a method that addresses the problem holistically, with a combination of statistical methods and logical formalisms. To that end, relevant work is drawn from a variety of areas to show this work's connection to broad but related themes of generalization that it seeks to unite. Thus, the following sections should be regarded with the long-term goal of achieving more general, adaptable agents capable of learning and using abstract representations of knowledge.

3.1 State Abstraction

The challenge of scaling of algorithms for large RL domains (where the set of possible states is extremely large or unbounded) is an active area of current research. Following from the theory of Li et al., exact state abstraction is categorized as one of five classes of aggregation at varying levels of granularity: model-irrelevance, Q^{π}-irrelevance, Q^{*}-irrelevance, optimal action irrelevance, and pi^{*}-irrelevance [92]. The purpose of grouping states together is to reduce the size of the state space while still abiding by the global reward and transition probabilities. The net result of abstraction is that unwieldy problems

are made more tractable, and it was proven that Q-function convergence holds for some of the finer classes of abstraction, and that coarser ones still permit guarantees for efficient planning. Although abstraction may help, transfer of knowledge and experience is not its explicit end-goal. Abstraction techniques such as state aggregation are often combined to do hierarchical RL, where a goal is decomposed into smaller, repeatable subtasks.

One perspective on concepts is that they achieve both abstraction and generalization, as opposed to (or in addition to) state aggregation-based abstraction alone. Abstraction is a process that "changes the representation of an object by hiding or removing less critical details while preserving desirable properties. By definition, this implies loss of information," whereas generalization is a process that "defines similarities between objects" but also "does not affect the object's representation. By definition, this implies no loss of information" [125]. Related topics are domain reduction (grouping objects), hiding (removing objects), and co-domain hiding (selective attention), each of which concept-based abstraction performs at several progressive levels of granularity. Ponsen et al. take knowledge transfer as a form of generalization, citing the methods of Q-value reuse, fitted Q-iteration, and region transfer, and discussing representations for transfer including relational options, agent-spaces, and finite-state machine-based strategies [125]. UCA_{GG} adaptive aggregation uses epsilon-bounded state aggregation (based on confidence estimate of regret). Ortner et al. discuss approximate abstractions, determining abstract states either by taking the average of transition probabilities and of rewards of ground states, or by using an exemplar ground state as reference [117]. Proximity-based non-uniform abstractions for approximate planning offer a significant alternative in using non-uniform "worldview" states (as opposed to uniform state aggregation). Worldviews work on a

case-by-case basis for each dimension, either leaving the feature concrete or abstracting (removing) it. Baum et al. introduce the Ostrich effect where an agent over-estimates a value by wrongly incorporating good information in a region (or vice versa), which could affect many forms of generalization and transfer, including concept-based ones [19].

3.2 Transfer & Lifelong Learning

Transfer learning concerns generalization of machine learning, where knowledge or experience gained solving one *source* task is transferred to a new (often related) *target* task. In this dissertation, I consider RL and planning in particular, where adapting to novel domains often means transferring skills, beliefs, or other representations of knowledge. Transfer learning under those paradigms centers on reusing behaviors from some solved MDPs to initiate good prior beliefs and speed learning in new domains. Examples of reusable knowledge include policies, partial policies [121], discovered options (temporally extended actions) [23, 159], and action priors [2].

Several subproblems were identified in a survey [154] for the general goal of transfer learning in RL which most approaches must tackle to some degree. One issue is that of *allowed task differences*, or the question of by how much the agent, number of objects in the state, and parameters of the task may vary among sources and targets. *Source task selection* is the subproblem of finding and applying an existing relevant source. A related issue is *task mapping*, determining how to map actions or other MDP components from one to another. There is also the question of *transferred knowledge*, or picking the representation and granularity of knowledge to store and apply (transition tuples, options,

shaped reward structure). Another question is that of *allowed learners* to compare against, as the knowledge used by TD, policy search, case-based, or other methods can differ greatly. Finally, various *metrics* could be used to evaluate the performance of transfer learning. Proposed measurements for intra-domain transfer include *jumpstart* (increased initial performance), *asymptotic performance* (increased final performance), total reward, the transfer ratio of the area under learning curves, and the task learning time (i.e., when it reaches a threshold of performance) [154].

One significant challenge of transfer learning is the notion of *negative transfer*, where applying knowledge actually hinders the efficiency or quality of a solution. Conceptually, transfer learning inherently encodes bias toward certain behaviors or beliefs. With this knowledge, the agent expects the world to be a certain way, and should reality defy expectation, then an agent may take longer to find or converge to a solution than if it had no prior knowledge at all. Techniques to mitigate negative transfer often incorporate task structure to determine how to perform no worse than an agent starting from an uniformed state.

Under the umbrella of transfer learning reside the subproblems of learning from demonstration and *lifelong learning*. Lifelong learning is the challenge of solving a sequence of tasks by one agent over an extended lifespan, as opposed to training one agent to one task. This goal is clearly relevant to robotics and more general AI, where it is too costly and inefficient to train an agent and then dispose of the knowledge gained. In the larger context of this dissertation, I am seeking to make agents build on experience in new environments; anomaly reasoning through concept formation aims for this directly.

Recent work has argued for more interpretable transfer in RL, leveraging an object-

based framework similar to OO-MDPs and using similarities to perform mappings from one set of object classes to another [129]. Another direction has focused more on advice frameworks, where learning is transferred in a multi-agent learning scenario, or from teachers to students [27, 29, 30]. Object-Focused Q-Learning (OFQL), while similar to the definition of OO-MDPs, differs in that each pairing of object-agent is treated as its own sub-MDP, with managed action risk. Q-values are computed for each pairing and a global Q-function estimate is taken from the maximum Q-value across the object pairs. Cobo et al. prove convergence to an optimal solution [26]. In a manner similar to OFQL, $\text{CLASS}_{\text{Q-L}}$ uses abstraction over object classes to compute Q-values. It uses two-phase learning, first executing an episode to completion, then performing a batch update of the Q-table; learning is generalized over the abstract classes [66].

3.3 Concept Formation

Concepts as a basis for knowledge, representation, and reasoning have a long history of study in the field of artificial intelligence [42]. Concept formation and learning are based in cognition and psychological models of human understanding. Concepts themselves vary in definition but usually hold to the idea of describing a set of attributes, features, or rules that are true for all members in a related set of examples called objects, instances, or entities.

At its fundamental level, concept formation is the idea of intelligent processing of percepts (input to sensors) to concepts (abstract representations) to actions (output from actuators). This description is purposefully general; it can be applied and used in various

types of settings, including supervised learning, clustering, case-based reasoning, planning and RL. Most commonly, observations from the environment are fed into a learning algorithm which is combined with a knowledge base. Input percepts are interpreted as concepts through their membership with certain (potentially dynamic, probabilistic, hierarchical) categories or relations. Crucially, learning is incremental. Concepts are created, grow, split, and change as learners make new observations. Thus, well-studied concept learning algorithms include decision trees (ID5, EPAM, Cyrus) and probabilistic clustering (COBWEB). Various schemes have been used for knowledge bases, including rules, formal logic, and the structures produced by learners themselves (as with the categories identified by COBWEB) [42].

From psychology, there are three views of concepts: axiomatic, prototype, and exemplar ones [106]. For artificial intelligence algorithms, axiomatic concepts are defined by a set of formal conditions (having to satisfy necessity and sufficiency). Prototype concepts are less restrictive, only specifying properties that need to be satisfied (a ball will have a shape, color, and material). Exemplar concepts are abstractions demonstrated by archetypal examples – an algorithm that learns to classify different dog species is identifying those categories as exemplary concepts. The generalized context model is an implementation of exemplars in memory, with similarity being a distance measure in psychological (concept feature) space.

Objects can be ordered, compared, and grouped by their concepts in an abstract space. One approach to this abstraction is a "version space" of hypotheses (concepts or constraints that apply to groups of objects). Version spaces are used to transfer from models at an object level to a more general model, incrementally learning concepts organized

into a hierarchy of specificity. That is, the specific model is generalized from positive instances, while the general model is specialized from counterexamples, until a bridge is found between them. Version spaces can aid classification of unseen data and provide an inductive leap that helps bridge machine learning with more formal, logic-based reasoning [112].

Recent work in concept formation has focused on leveraging neural architectures to extract features and perform one-shot learning. Where traditional classification takes many examples to learn a few classes, one-shot learning is the challenge of learning many classes without supervision from few examples. This goal aligns with human learning better than standard supervised learning. For example, it would not take many examples for someone who has never seen a banana before to learn that concept and apply it to any new example they may encounter, but most machine learning algorithms would need hundreds to thousands of examples. One paper achieved human-level learning of concepts on Omniglot, a complex data set consisting of thousands of types of handwritten characters (some real, some artificial) for which there are only tens of examples for any given character. They used Bayesian program induction to capture the structural composition of visual concepts (parsing the direction of strokes and location of joints). Additionally, since their technique captures a joint probability, they model concept membership probabilistically and can generate valid, novel examples of the learned classes [86].

Another approach to incremental human concept learning, TRESTLE, takes a more compositional approach to concepts, performing feature extraction to parse a scene and determine the relational structure of objects. TRESTLE hierarchically organizes probabilistic concepts that maintain the counts of member objects and attributes. The main al-

gorithm for structuring the hierarchy is COBWEB, for which a categorization tree is built (by creating, updating, merging, and splitting concept nodes) using the category utility of concepts given their parents. TRESTLE was evaluated in both a supervised and an unsupervised setting and approximated human-level performance at identifying and grouping concepts for RumbleBlocks, an interactive physics-based construction game [99]. TRESTLE compares its evaluation with a related approach, conceptual feature extraction (CFE), that discretizes state space, generates a grammar of components and their relations, and parses the discretization into a feature vector [58].

3.4 Memory

Representing and maintaining memory for sequential decision making and other temporal learning problems is an open challenge. In brief, memory is the capacity to read and write information over time. Memory can serve either as a more long-term knowledge base (for instance, of learned concepts), or to preserve a more short-term notion of the immediately preceding states, context, and surrounding. Addresses are a common feature of memory, where an address is an index to a specific entity in memory. Notions of locality and spatiality are also included in some models of memory (such as the position of a Turing machine's read-write head along a tape). Associated memories, for instance, can be found in a local neighborhood of memory space, or contain address references to similar memories. Memory has a long history of study in the fields of computer science, artificial intelligence, and cognitive science [61].

One mechanism for recording experiences (including learned concepts) is *sparse*

distributed memory (SDM) [73]. SDM acts as a random-access memory that has three registers, one each for addressing, reading, and writing. Percepts are encoded as words and stored in counters addressed in an expansive binary space (memory). Addresses, point locations in memory, are sparse and distributed to provide coverage. Associated memories can be measured as lying within some given Hamming distance radius. When input percepts are written to memory, they are indexed to a point in memory space but not recorded there exactly. Instead, all addresses are considered, and any within a given distance threshold to the indexed point are activated. The data counters at these indexed memories are incremented. Hence, observations are distributed over regions of memory (stored sparsely). Reading from memory takes a sum over the sparse neighborhood in memory space. If the sum of activations passes a gating threshold, non-zero output is passed to an out register. In effect, SDM learns an encoding and reconstruction of input, similar to neural networks such as autoencoders.

SDM has been implemented as a function approximation technique more commonly referred to as "Kanerva coding" [172]. As differentiated from tile coding and basis function-based function approximation, SDM scales well since its computational complexity depends only on the number of addresses in memory, and not the size of the state-action space. Weight parameters are learned for addressing; linear value function approximation simply takes the sum of weighted activated addresses.

Recently, research into deep neural architectures has investigated the inclusion of a memory storage component while learning. Examples include neural Turing machines (NTM), differentiable neural computers (DNC), memory-augmented neural networks (MANN), and recurrent entity networks (EntNet). The NTM recreates a Turing machine via a neural

network and includes readable and writable memory [56]. A DNC [57] extends NTMs with a recurrent network controller that governs the contents of a write head and two read heads that interact with a separate, external (differentiable) memory component that also has temporal links to preserve a kind of state of recent updates. The DNC architecture was able to learn arbitrary graph structures, including the London Metro, and answer questions about how to traverse from one station to another. In an RL setting, a DNC was able to solve the classic "blocksworld" stacking task, learning to encode temporal subgoals into memory. MANNs tackle generalization by considering the one-shot learning case, where memory is, again, an external module that is written to and read from [136]. EntNets maintain a dynamic, gated memory structure that has its own trainable parameters; its success on entity resolution in question-answering domains indicates that parameterized memory storage can improve tracking complex world states [63].

3.5 Case-Based & Analogical Reasoning

Case-based reasoning considers memory as a set of learned "cases" that may be applied and adapted to a new problem being faced. A typical case-base reasoning workflow, given some new problem, is the sequence of processes: retrieval, adaptation, evaluation, and storage. Adaptation fits previous cases or rules, through the application of a model or heuristics, to the new problem. For evaluation, the proposed solution must be simulated or executed, recording the results. Once the new problem is solved, it is stored as a case. Storage also concerns determining proper indexing schemes for fast retrieval of cases. Failures might also be stored to help recognize such cases in the future.

Analogical reasoning, as a subgoal of case-based reasoning, considers the correspondence problem, or mapping from a source entity to a target. It performs transfer learning by selecting known cases (or objects), and can employ concept formation or other representations to express similarity. SDM is an example of a memory formalization that can achieve generalization through learned analogical mappings [37].

3.6 Anomaly Detection

Anomaly detection focuses on the identification of outliers or patterns of data that are unusual with respect to expected observations. Anomalies as defined for this work are a subset of this more general definition. Commonly, input data is formulated as objects (instances, rows) with attributes (features, columns). From a given data set, an algorithm seeks to find anomalies described by one of three categories. *Point anomalies* are anomalous objects with respect to an entire data set, and they can often be identified as residing in a sparse or low-populated region of feature space. *Contextual anomalies* are objects that are anomalous only under certain circumstances, called a context, which is most often used for time-series data, or domains where the neighborhood of feature space around an object should be considered in determining its degree of normality. *Collective anomalies* are sets of objects that, together, correspond to an anomalous event with respect to other data. For example, some word bi-grams may not be unusual when observed individually, but their co-occurrence may be anomalous. Anomaly detection is broadly applicable to a variety of fields including medicine, text and video analysis, intrusion and fraud detection, and financial forecasting [25].

3.7 Concept-Based Learning

Combining concept learning in a decision-making context has been explored in recent papers. One approach examined concept discovery in a robotics scenario using incremental graph-based object-relation extraction to create structural concepts. Learned structures were combinable graphs of objects and their relations such as "object-in-room" or "behind-door" for a robot exploring an office environment. The result is that a reinforcement learning agent is able to find simple and hierarchical concepts to better explore and transfer knowledge gained from solving tasks [156]. Another approach focused more on learning functional concepts for knowledge transfer in RL. Samples from perceptions map to points in an agent's "function space." Functional concepts, express clusters of these points that influence an agent's actions. For transfer, a source agent converts vectors of Q-values into functional concepts (clusters) that a target agent can use to speed learning [115].

Chapter 4: Reasoning about Anomalies

4.1 A Framework for Anomaly Reasoning

I define *anomaly reasoning* as the ability to recognize and react appropriately to unfamiliar objects, with respect to an agent and its environment. Though broad in scope, I decompose anomaly reasoning into three distinct, successive parts: the *identification* of an anomaly, *interpretation* of it, and the *adaptation* to it. Identification encompasses extracting features and segmenting objects from perceptions to detect any anomaly. Interpretation projects an identified anomaly into a (potentially learned) internal conceptual model of the world. Finally, adaptation determines which aspects of the interpreted anomaly are most relevant to the agent's current context and what subsequent decisions are most suitable. While solving anomaly reasoning altogether is a difficult though worthwhile aim, I take each subproblem individually, to be addressed separately, and ultimately integrated into a more cohesive pipeline. Such a framework is summarized in Table 4.1.

4.1.1 Identification

The process of anomaly identification consists of parsing objects from observations and using domain knowledge to find anomalies in this context. The input is a vector of percep-

1. **Identification**
   - $\textsc{Recognize}(\vec{x}_t) \rightarrow \mathcal{O}_t$
   - $\textsc{Detect}(K, \mathcal{O}_t) \rightarrow \mathcal{O}_t^A$

2. **Interpretation**
   - $\textsc{Conceptualize}(\mathcal{O}_t) \rightarrow \mathfrak{B}_t$
   - $\textsc{Store}(K, \mathfrak{B}_t) \rightarrow K$
   - $\textsc{Abstract}(\vec{a}, K) \rightarrow \mathfrak{C}_t^{\vec{a}}$

3. **Adaptation**
   - $\textsc{Highlight}(K, \mathfrak{C}_t^{\vec{a}}, \mathcal{O}_t) \rightarrow \vec{h}_t^{\vec{a}}$
   - $\textsc{Emphasize}(\vec{a}, \mathfrak{C}_t^{\vec{a}}, \vec{h}_t^{\vec{a}}) \rightarrow \vec{e}_t^{\vec{a}}$
   - $\textsc{Adapt}(K, \vec{a}, \mathfrak{C}_t^{\vec{a}}, \vec{h}_t^{\vec{a}}, \vec{e}_t^{\vec{a}}, \mathcal{O}_t) \rightarrow \alpha$
   - From decision based on α, obtain resulting feedback ρ (error, reward)
   - $\textsc{Update}(K, \alpha, \rho) \rightarrow K$

▶ Input: perceptions, $\vec{x}_t$, knowledge K

▷ the set of objects (context)

▷ the anomalies, a subset of objects

▶ Input: objects $\mathcal{O}_t$, anomalies $\mathcal{O}_t^A$

▷ the set of concepts (a lattice)

▷ expanded domain knowledge

▷ the set of concepts for anomaly $\vec{a}$

▶ Input: concepts from all anomalies, $\mathfrak{C}_t^A$

▷ a vector of concept weights

▷ a vector of attribute-value weights

▷ the adaptation, an abstract state, action, or reshaping of perceptions that reduces uncertainty

▷ updated knowledge, based on the adaptation and the decision's result

Table 4.1: Summary of a pipeline for anomaly reasoning.

tions $\vec{x}_t$ at time t along with some knowledge base K, and the output is a set of anomalous objects $\mathcal{O}_t^A$. In the language of anomaly detection literature (see Section 3.6), these objects are contextual (conditional) anomalies, with the current context $\mathcal{O}_t$.

The domain knowledge K is used by the agent to interpret anomalies. K could be static (for instance, if dealing with anomalies in text, K may be an ontology like WordNet) or learned incrementally as the agent makes decisions and observes new states, which is how the solution is framed, storing and updating behavioral information at concept in a hierarchy. For this implementation, K is a sparse memory containing concept lattices that are abstracted and stores behavioral information indexed by binary vectors of concept intents (the encoding of attribute-value sets).

4.1.1.1 Object Recognition

In keeping with the principle of a general framework that could potentially be implemented in various ways, I explicitly include recognizing objects as the first step in identifying anomalies. Anomaly reasoning makes only the assumptions that agents receive observations of their environment, irrespective of the form those perceptions may take, and that the agent can convert those observations into object-based representations. The input is a vector of perceptions at the current time, $\vec{x}_t$, and the output is a set of objects, $\mathcal{O}_t$.

From the true state of the world s_t at time t (as in an MDP environment), I define the agent's observation of s_t as a vector of n features $\vec{x}_t = \langle x_1, x_2, \ldots, x_n \rangle$. Observations are then processed through an object recognition function into a set of objects, RECOG-

NIZE$(\vec{x}_t) \rightarrow \mathcal{O}_t$.

Following from the OO-MDP formulation, states are sets of instantiated objects where an object is a list of assignments to attributes called attribute-values. Without loss of generality, all objects can belong to the same class C consisting of the full space of m attributes, $C = \langle C.a_1, C.a_2, \ldots, C.a_m \rangle$. Every attribute has its own domain of possible attribute-values, $Dom(C.a_j)$, with some special "null" attribute-value $\varnothing$ that denotes the absence of the attribute for the given object. Thus, each object $\vec{o}_i \in \mathcal{O}_t$ is a vector of instantiated attribute-values, $\vec{o}_i = \langle v_{i,1}, v_{i,2}, \ldots, v_{i,m} \rangle$.

The entire set of objects can be represented as a matrix

$$\mathcal{O}_t = (v_{i,j}) = \begin{pmatrix} v_{1,1} & v_{1,2} & \cdots & v_{1,m} \\ v_{2,1} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ v_{n,1} & \cdots & \cdots & v_{n,m} \end{pmatrix},$$

such that $v_{i,j}$ is the attribute-value of attribute c_j for object $\vec{o}_i$. An OO-MDP state maps directly into this representation. Simultaneously, $\mathcal{O}_t$ can easily be binarized and serve as a formal context from which a concept lattice can be constructed.

To apply FCA, let us define $\mathcal{M}_t$ as the set of all μ_t possible attribute-values ever observed up to time t, so $\mathcal{M}_t = \langle v_1, v_2, \ldots, v_{\mu_t} \rangle$. The object set is re-represented as a formal context (binary matrix) $\mathcal{O}_t = (b_{i,k})$ such that $b_{i,k} \in \{0, 1\}$ and $b_{i,k} = 1$ when object $\vec{o}_i$ has an attribute set to attribute-value v_k, and $b_{i,k} = 0$ when the object either lacks the attribute or the relevant attribute is not set to v_k. To capture uncertainty, $b_{i,k} = 0$ is also

used when the presence of v_k is unknown. With this interpretation of an OO-MDP, the incidence of objects and attribute-values is easily induced.

Note that this formulation can grow dynamically and is robust to new objects and attribute-values. A canonical ordering M_t of all observed attribute-values up to time t is preserved. For instance, if a new attribute-value v is observed at time t, it is appended to M_t (in parallel, $\mathcal{M}_t = \mathcal{M}_{t-1} \cup \{v\}$). Then $\mathcal{O}_t$ will simply have an additional column. If necessary, all past $\mathcal{O}$ matrices could be updated immediately to this new attribute-value space by appending a column of zeros, which is valid since no object among those contexts ever possessed v.

This definition is left purposefully general, since the features may vary widely based on the domain. For instance, if $\vec{x}_t$ is an image, then RECOGNIZE may be an image segmentation algorithm, each $\vec{o}$ would be a region or segmented object, and attributes could be any recognized semantic properties (color, material, shape) or simply activation levels from features extracted by a neural network. For a text corpus, if $\vec{x}_t$ is a document, then RECOGNIZE could be a topic modeling algorithm that extracts n-grams from a bag-of-words model to function as the attributes for each recognized topic, $\vec{o}$. Alternatively, RECOGNIZE could find anomalous named entities or events among text passages.

4.1.1.2 Anomaly Detection

The next procedure is to detect anomalies among the object set given the agent's domain knowledge. This step is expressed as $\textsc{Detect}(K, \mathcal{O}_t) \rightarrow \mathcal{O}_t^A$, with $\mathcal{O}_t^A \subset \mathcal{O}_t$ and K, a knowledge base. Each anomaly $\vec{a} \in \mathcal{O}_t^A$ is an object detected as anomalous given the

current context of objects and existing domain knowledge. K may be derived from the concept model learned over time by the framework (described in Section 4.1.2). The K is left general here; some approaches to anomaly detection use an alternative, domain-specific ontology.

For domains with no prior information given, such as for a planning or reinforcement learning agent, every object may be considered anomalous until it reaches some threshold beyond which it is considered "known." This threshold may be described probabilistically and compositionally in terms of concepts that have encountered and stored in memory frequently.

4.1.2 Interpretation

For interpretation, anomalies are considered in context, mapped into abstract representations in conceptual space. The input is the context at the current time and the set of detected anomalous objects, $\mathcal{O}_t$ and $\mathcal{O}_t^A$ respectively. The output given the context is, for each anomaly $\vec{a} \in \mathcal{O}_t^A$, the set $\mathfrak{C}_t^{\vec{a}}$ of concepts formed from the anomaly. Each $\vec{a}$, interpreted as a set of concepts, is passed on to the following process which selects an adaptation given all detected and interpreted anomalies.

4.1.2.1 Concept Formation

From the given context $\mathcal{O}_t$, concepts are formed and stored in a structured representation by $\textsc{Conceptualize}(\mathcal{O}_t) \rightarrow \mathfrak{B}_t$, yielding a lattice that is a subset of all possible concepts, $\mathfrak{B}_t \subset \mathfrak{C}$. Each $c \in \mathfrak{B}_t$ is a concept defined generally (different representations

of concepts could be used), and $\mathfrak{B}_t$ captures some graphical or relational information among them. For the remainder of this work I consider the FCA formulation of formal contexts and concepts (as described in Section 2.3); however, other encodings or representations for concepts could be used. CONCEPTUALIZE is implemented as some concept-mining algorithm, and $\mathfrak{B}_t$ is a concept lattice derived from $\mathcal{O}_t$, or in FCA parlance, the lattice is the relation $\mathfrak{B}_t(\mathcal{O}_t, \mathcal{M}_t, \mathrm{I}_t)$ where $\mathcal{M}_t$ is the set of all attribute-values and $\mathrm{I}_t : \mathcal{O}_t \times \mathcal{M}_t \to \{0, 1\}$ is the incidence $\mathrm{I}_t(\vec{o}_i, v_k) = 1$ when object $\vec{o}_i \in \mathcal{O}_t$ possesses an attribute currently equal to attribute-value $v_k \in \mathcal{M}_t$.

With $\mathfrak{B}_t$, a set of concepts is obtained: $\mathfrak{B}_t = \{(A, B) \mid A \subseteq \mathcal{O}_t, B \subseteq \mathcal{M}_t, A = B^{\downarrow} \text{ and } B = A^{\uparrow}\}$. Optionally, a domain model K is updated and retained, $\textsc{Store}(K, \mathfrak{B}_t) \to K$. As with many implementation choices, there are multiple ways K might be implemented. One such approach is an iteratively abstracted set of attribute sets derived from $\mathfrak{B}_t$, an example of which is further described in Section 4.2.1. Another way is to record object counts, which is how concept-aware QL works (Section 5.3.1).

In the case that DETECT from Section 4.1.1.2 uses concepts to detect anomalies, the CONCEPTUALIZE and STORE functions could instead be employed at that point, rather than here as a separate, subsequent step.

4.1.2.2 Anomaly Abstraction

Each anomaly $\vec{a}$ is abstracted to the subset of formal concepts extracted from the context for which it is a member, $\textsc{Abstract}(\vec{a}, K) \to \mathfrak{C}_t^{\vec{a}}$, with $\mathfrak{C}_t^{\vec{a}} \subseteq \mathfrak{B}_t$ or $\mathfrak{C}_t^{\vec{a}} \subset K$. In other words, $\mathfrak{C}_t^{\vec{a}}$ are the concepts formed by observing the anomaly in context, with respect to

the agent's prior knowledge base. For each concept $c \in \mathfrak{C}_t^{\vec{a}}$, it is a pair of sets $c = (A \subset \mathcal{O}_t, B \subset \mathcal{M}_t)$ such that $\vec{a} \in A$ and $\forall\, v \in B, v \in \vec{a}$. That is, an anomaly is in all object sets of its concepts, and all attribute-values from its concepts are found in the anomaly. Taken together, after all anomalies are abstracted, there is $\mathfrak{C}_t^A = \{\mathfrak{C}_t^{\vec{a}}\}$, the set of the concepts formed for each of them.

All detected anomalies are now interpreted as a set of concepts, some of which may be novel, and these new perceptions are incorporated in the domain knowledge K.

4.1.3 Adaptation

The next phase of anomaly reasoning is to take interpreted anomalies and produce either a behavior or a re-representation of observed state that reduces uncertainty. The input consists of $\mathfrak{C}_t^A$, the set of concept sets formed for each detected anomaly, while the output is an adaptation to the anomaly, α, to be used by the agent or system implementing the anomaly reasoning framework.

4.1.3.1 Attribute Highlighting

To adapt, an agent first highlights the concepts crucial to its decision making. Thus, an agent makes use of its domain knowledge K in an algorithm to highlight the relevant concepts from the interpreted anomaly $\mathfrak{C}_t^{\vec{a}} \in \mathfrak{C}_t^A$. This procedure takes the form of a function applied to each anomaly, $\textsc{Highlight}(K, \mathfrak{C}_t^{\vec{a}}, \mathcal{O}_t) \rightarrow \vec{h}_t^{\vec{a}}$, where the relevancy weight vector $\vec{h}_t^{\vec{a}} = \langle h_1, h_2, \ldots, h_d \rangle$ has one weight for each d concept in $\mathfrak{C}_t^{\vec{a}}$. With $\vec{h}_t^{\vec{a}}$, the agent can explicitly reason about which concepts are irrelevant or most relevant given

the context $\mathcal{O}_t$. Moreover, since the highlighting for each interpreted anomaly is done using the same K and context, elements of the $\vec{h}_t$ for different anomalies are on the same scale and readily comparable.

In most cases it is likely that $\vec{h}_t^{\vec{a}}$ must be learned from feedback based on the agent's subsequent decisions. As an example, the concept-aware algorithms described in Section 5.3.1, which use concepts as features for function approximation, treat $\vec{h}_t^{\vec{a}}$ as the weight parameters and learn them through Bellman backups following some bootstrapped temporal difference algorithm such as SARSA.

Optionally, an additional function could be used to achieve more granular understanding of relevancy, at the attribute level. Consider EMPHASIZE$(\vec{a}, \mathfrak{C}_t^{\vec{a}}, \vec{h}_t^{\vec{a}}) \rightarrow \vec{e}_t^{\vec{a}}$, which takes the anomaly and its highlighted concepts to find an attribute-wise relevancy weight vector, $\vec{e}_t^{\vec{a}} = \langle e_1, e_2, \ldots, e_m \rangle$. That is, for each m attribute in the class of the object, the current attribute-value can also be re-weighted based on what concepts were found to be relevant and then used in later decision making. As with $\vec{h}_t^{\vec{a}}$, it is likely that $\vec{e}_t^{\vec{a}}$ would need to be learned with feedback from decisions made based on the adaptation that $\vec{e}_t^{\vec{a}}$ suggested.

4.1.3.2 Anomaly Adaptation

Finally, the agent must use its knowledge and highlighted concepts to adapt to the anomalies. For each anomaly, the function ADAPT$(K, \vec{a}, \mathfrak{C}_t^{\vec{a}}, \vec{h}_t^{\vec{a}}, \vec{e}_t^{\vec{a}}, \mathcal{O}_t) \rightarrow \alpha$, where α is some representation digestible by the agent using the anomaly reasoning framework that reduces uncertainty in decision making. This definition is left general as α could take

various different forms, especially depending on what K contains. For instance, α could simply be an action that the agent should take, which would assume K then stores and updates information that directly relates concepts to actions (this is how concept-aware algorithms work). Or, α could be a replacement of the original anomalous object with "filled in" attribute-values for noisy or unobserved attributes. More broadly, $alpha$ could be a remapping of the original input vector of observations $\vec{x}$ to a vector $\hat{\vec{x}}$ with reduced uncertainty. A naive version of ADAPT would be to map each anomaly to its closest analogue in terms of concept similarity. More sophisticated approaches would directly consider the current context to determine holistically what concepts and attribute-values are relevant. Note that I define α over the anomalies, such that each anomaly recommends its own adaptation. If necessary, various schemes could be used to select one α among the set of produced from all anomalies, for instance based on the highlight or emphasis vectors. Alternatively, they could be collected and applied together as a single adaptation, for instance by replacing each anomalous object or uncertain attribute-values with a known analogue or predicted values.

To have the anomaly reasoning framework improve domain knowledge and learn over time, some mechanism must be included to transfer feedback (error, loss, reward) based on the effects of using α as the selected adaption. Assuming the agent makes a decision based on α, the resulting ρ signal should be used to align domain knowledge with new information. I refer to this step by the procedure UPDATE(K, α, ρ). This step is similar to the STORE function of Section 4.1.2.1, when the agent perceives how the concept lattice $\mathfrak{B}_{t+1}$ differs from $\mathfrak{B}_t$. For instance, the concept-aware algorithms of Section 5.3.1 implement ADAPT by incorporating a learned policy and taking an action α. Then, UP-

DATE is based on the reward signal $\rho = r_{t+1}$ from the transition $(s_t, \alpha_t, r_{t+1}, s_{t+1})$ a parameter-wise gradient descent update improves the concept-based feature weights that serve as K.

With the completion of the anomaly reasoning pipeline, the agent is able to take perceptions, build and update a model of its world, and produce an interpretation that guides behavior to reduce uncertainty, allowing it to generalize in a greater variety of environments.

4.1.4 Properties

In general, I make use of FCA as it provides a good balance between hierarchical, symbolic reasoning and low-level encoding, while granting several other advantageous properties. First, objects and concepts alike can be represented dually and ordered by their respective intent, which corresponds to a binary vector (or bitstring). This benefit also provides an extensible abstract space that simplifies measuring similarity, facilitating the search for relevant or analogous concepts, objects, and attributes. Second, formal concepts often yield groupings of objects that human observers find semantically meaningful, for instance concepts that capture all objects of a certain shape or color. This ability can lead to greater explainability, where an agent can point directly to the concepts most relevant to its decision (in relation to the other concepts present) as justification. Third, formal concepts derived from OO-MDP states represent useful semantic sub-states that recur across domains, allowing the agents to consider partitions of state space where the formal concepts are found. For example, concepts which occur in all states can be iso-

lated as irrelevant to solving the given MDP. Additionally, conceptual sub-states could be clustered to find topological properties and examine how a value or action-value varies across those dimensions. As noted in Section 4.1.2, definitions of concepts other than FCA could be used instead. However, to suffice for anomaly reasoning, they would at least need to afford these properties to achieve anomaly reasoning.

4.1.4.1 Representation and Indexing

Objects can be stored as binary vectors, or unsigned bitstrings where activated bits indicate the presence of a specific attribute-value and unset bits indicate the lack of that attribute-value according to the canonical ordering M_t. Likewise, the attribute-value set of a concept (corresponding to its intent) may also be stored as a bitstring, again with "on" bits denoting membership of that attribute-values in that concept. Specifically, for an object g or a concept $c = (A, B)$, its intent is $\{g\}^{\uparrow}$ or B, respectively. Any intent B consists of a subset of known attribute-values $m \in B \subset \mathcal{M}_t$ and can be ordered according to the canonical ordering M_t of attribute-values (which grows as new ones are observed). Thus, a binary vector $\vec{b}$ can be derived for intent B, $|\vec{b}| = |M_t|$, such that $\forall\ m_i \in M_t$, each element in $\vec{b}$ is defined:

$$b_i = \begin{cases} 1, & \text{if } m_i \in B \\ 0, & \text{otherwise} \end{cases}. \tag{4.1}$$

Note that the binary vector of an intent $\vec{b}$ can be translated immediately into an unsigned bitstring. For convenience and extensibility, these strings are considered "little-endian," with $1_2 = 1$, $01_2 = 2$, $001_2 = 4$, and so on. This choice means that when

a new attribute-value v_k is observed and $|M_t|$ grows larger, the bitstring representations of intents from concepts seen at any time before t do not need to change (their bitstring would simply have a "0" at index k). Thus, the old ones are still valid since no previous object or concept ever possessed the newly observed attribute-value.

I refer to an object's or concept's *intent index* as the decimal natural number to which its intent's bitstring is equal. A concept of no attribute-values (empty intent) has an intent index of 0. A concept with all attribute-values observed up to t has an intent index of $|M_t|$.

4.1.4.2 Similarity Metrics

With a binary representation and the intent indexing, all objects and concepts can be measured in the same attribute-value (intent) space, allowing us to make object-to-object, object-to-concept, and concept-to-concept comparisons. This property is welcome because it expands the ways in which I can relate objects and concepts, in terms of set membership and not just as vectors in some high-dimensional space. A standard approach to measuring similarity among objects (rows or vectors in a data set) might consist simply of computing the cosine similarity or Euclidean distance, or performing dimensionality reduction (e.g., principal components analysis) to obtain a compressed representation and applying those same metrics. However, these techniques lack the main benefit granted by binary representation: set membership. That is, it is straightforward to find the concept sets to which an object belongs, which concepts are shared between two objects, and the overlap of shared attributes between two concepts, all using bitwise operators. For in-

stance, if an object possesses all the on-bits found in a concept, then it must belong to that concept's extent. Similarly, the super- and sub- concept relations are apparent, and similarity among concepts can be assessed by considering how many bits (on, off, or both) on which they overlap.

Thus, for objects and concepts, various types of set-similarity (or distance) metrics are applicable, and I note four: Hamming, Euclidean, Jaccard, and Rogers-Tanimoto [175]. The Hamming distance for two bitstrings $\vec{x}$ and $\vec{y}$ with magnitude $B = |\vec{x}| = |\vec{y}|$ is simply the number of bits on which two strings differ (i.e., the cardinality of their XOR), while Euclidean distance is the square root of the Hamming distance. Set similarity based on Euclidean and Hamming distances is just $sim_{\text{distance}} = 1 - \frac{\text{distance}}{|B|}$. It is significant that Hamming captures the complete distance among strings, such that every bit that differs increases the distance. For example, suppose there are two object o_a and o_b, and a concept c. Further suppose that the objects are identical except for their color (o_a is red, o_b is blue), and that c contains all attribute-values of o_a and o_b except any related to color. Then under Hamming distance, c is more similar to both o_a and o_b (only one bit is different) than o_a and o_b are to each other (they differ by two bits). Thus, Hamming distance is a useful tool for assessing similarity when a greater granularity of object-concept and object-object differences is required.

Jaccard set similarity is the cardinality of their intersection divided by the cardinality of their union, $sim_J = |\vec{x} \cap \vec{y}|/|\vec{x} \cup \vec{y}|$. Rogers-Tanimoto set similarity is the count of shared bits over that same number plus two times the number different, or $sim_{RT} = |\sim(\vec{x} \oplus \vec{y})| \ / \ \big(|\sim(\vec{x} \oplus \vec{y})| + 2|\vec{x} \oplus \vec{y}|\big)$. While there is no gold standard of accuracy, the Rogers-Tanimoto metric has been recommended as having the best dis-

criminative power in terms of entropy variations and recognition accuracy on various tasks [175]. In particular, Rogers-Tanimoto doubles the count of bits in the XOR between two bitstrings. This property makes more semantic sense for concepts than the Sokal-Sneath metric (which doubles the similarity), since concept bitstrings are sparse (mostly off-bits). One perspective on Rogers-Tanimoto is that the numerator is the number of bits in which the bitstrings agree, and the denominator is the number of all "realized" bit states [122].

4.1.4.3 Object Hierarchies

A crucial benefit of FCA is that the concept lattice inherently captures abstract groupings of objects that correspond to classes or categories to which that object would belong. While I do yet not describe leveraging object hierarchies directly in anomaly reasoning, they become useful in subsequent work, especially for transferring behaviors by considering what has been learned for objects in both super-concepts and sub-concepts.

4.1.4.4 Hypothetical States

Formal concepts inherently capture semantic, abstract groupings of features of a given context. When this context is drawn from the state space of an MDP, the concept-based features correspond to abstract substates that describe meaningful properties of state space. For instance, a substate derived from a concept may contain all red block-shaped objects, or gray door-shaped objects. The vast majorities of these sub-states are not reachable via any transitions since they do not literally exist in any ground state. Each

concept can be used to partition state space by either presence or lack of the concepts; literal ground states are conjunctions of these concept-based substates. They perform domain reduction (grouping objects), hiding (removing objects), and co-domain hiding (selective attention), each of which is a valuable form of abstraction [125]. Thus, by thinking of states in terms of concept presence, an agent could construct new hypothetical states through novel combinations of concepts. These "imagined" states could be evaluated on the probability of their existence and their potential value (equivalently, the action-value of certain actions taken in when in them). Hypothetical states could serve as goals for a reasoning agent to posit and then plan to reach, which would be especially useful in partially observable and more uncertain domains. Similarly, training an agent to solve parameterized tasks with concept-based task descriptors could help attain an even higher-level, general ability to transfer skills across related tasks, including those not seen before.

4.2 Interpretation as Classification

The initial approach centers on analyzing the utility of FCA for anomaly interpretation and adaptation. To this end, I have focused on developing novel methods with the hypothesis that anomaly reasoning necessitates combining a machine learning approach with an organized representation of knowledge. The initial steps were first to develop a prototypical interpretation system that observes different contexts of objects and builds an abstract hierarchy of object classes while interpreting novel ones, and second to implement function approximation based on formal concepts to aid transfer learning for concept-aware

decision-making algorithms.

4.2.1 Interpretation Prototype

Ultimately, anomaly reasoning agents should subsume novelty quickly for complex domains rich in objects and attributes. One such domain is NetHack, a computer game first released in 1987. The environment in NetHack consists of partially observable "gridworld"-style levels populated by monsters that a player (agent) must traverse while attempting to survive. An agent completing successive levels will encounter many new objects (monsters, items, furniture) such that behavior learned in the past can transfer to these new entities.

My goal is to examine the effectiveness of FCA as the central vehicle for anomaly interpretation, separate from detecting and reacting to anomalies. Therefore, I assume an agent is given an object-based representation and, subsequent to interpretation, do not consider the adaptation or interaction, for instance with an MDP.

The interpretation prototype follows a training-testing pattern, first learning on a given context drawn randomly from a distribution of objects in a training set. When a context is observed, a concept lattice is built and then abstracted to a set of attribute sets (CONCEPTUALIZE). These sets are then added to the initially empty knowledge model K, a hierarchical graph of these attribute sets that grows as more concepts are discovered (STORE). Then, in a testing phase, the K is made static so it can be evaluated on anomalies, which consist of unseen objects (from a test set). Each anomaly is interpreted through K as a set of concepts (ABSTRACT). I consider the accuracy of the interpreta-

tion by various similarity measures, and report Rogers-Tanimoto similarity for concept bitstrings in subsequent discussions since this metric captures a better granularity of differences between binary sets, as explained in Section 4.1.4.2. Overall, the interpretation process offers the closest known analogue that the anomaly matches, which could theoretically be used as the output α for ADAPT.

To extract concepts and retrieve a lattice, I employ the IN-CLOSE2 algorithm, which follows the exploration strategy of Fast Close by One in conjunction with a test of canonicity to search more efficiently [7]. IN-CLOSE2 defines two subprocedures: a recursive concept builder and the canonicity test ISCANONICAL. Concepts are built by recursively computing extents, checking if an attribute is part of the extent and testing the intersection of a given concept's current extent with the extent of the next attribute. The latter is used whenever a new intersection is found to check if it corresponds to the extent of a new canonical concept or if it is already contained in any of the concepts already computed. If this candidate extent is canonical, it is added to the list of children of the current extent being constructed. Thus, the algorithm performs a kind of dual breadth-then-depth search, determining attributes that would belong to a concept (in queue order) and then expanding its child concepts (in stack order). A central challenge of FCA algorithms is to compute closures efficiently; IN-CLOSE2 minimizes the overhead of repeatedly computing closures by doing so implicitly with the limited backtracking that ISCANONICAL performs. The asymptotic worst-case time complexity of IN-CLOSE2 is $O(|\mathfrak{B}||\mathcal{G}||\mathcal{M}|^2)$ [164]. The actual construction of the lattice is done with Lindig's UPPERNEIGHBORS algorithm that constructs the lattice bottom-up, starting with the zero concept found by IN-CLOSE2, recursively finding the parent concepts of each given concept, with worst case complexity

of $O(|\mathfrak{B}||\mathcal{G}|^2|\mathcal{M}|)$ [93].

4.2.2 NetHack Monster Data Set

I create a data set consisting of all classes of mobile entities in NetHack called *the NetHack Monster data set* extracted from the game's data files and contains 390 object types (monsters), each with five observable attributes: shape, color, speed, alignment, sound, and size. I binarize the data yielding $\mathcal{M}$, the set of all possible attribute-values ($\mathcal{M} = 162$). Building a concept lattice on the full data set results in 1408 formal concepts. Visualizations of the data are presented in Figure 4.1, Figure 4.2, and Figure 4.3 to show that the observable features are sufficient to capture the species-like relations among monsters. Figure 4.1 shows a smaller subset of the NetHack monsters data set in a heatmap of Hamming distance with dendrogram showing the relationships of objects in the data. For instance, smaller creatures and animals group together (such as giantrat and newt), and are separated clearly from the larger humanoid cluster with the acolyte and elvenking. Figure 4.2 captures the full dendrogram of NetHack monsters, with an arbitrary threshold set at a distance of 0.055 to yield 22 clusters. For the most part, clusters separate monsters clearly along semantically sensible lines that parallel behaviors they may exhibit in the game. The large yellow cluster in Figures 4.2a and 4.2b contain humanoid monsters, while the red cluster in Figures 4.2b and 4.2c contains primarily animals or otherwise small monsters. These clusters indicate that analogies made among monsters by an agent playing NetHack would be useful, as almost any given monster has subsets of others that it is related to at varying degrees. The t-distributed stochastic neighbor embed-

ding (t-SNE) of the NetHack Monsters data set in Figure 4.3 shows the relative positions of clusters from the high-dimensional manifold projected down to a 2D space. For example, small and animal-like creatures are in the orange and yellow clusters in the lower left, while more friendly humanoids are in purple in the lower right, and more dangerous, complex monsters are projected to the center and top regions.

4.2.3 Lattice Abstraction

A parallel goal was to ensure that interpretation would scale. This is achieved in part by a process of abstracting the concepts that are formed and storing a more compressed representation. More specifically, a formal context in the FCA sense is fully recoverable from a concept lattice. Since the goal is to use lattices as domain knowledge K in anomaly interpretation, an agent could theoretically just keep all lattices ever constructed and perform, in essence, a kind of k-nearest neighbors algorithm. However, it is not preferable to retain all information ever derived from perceptions. Storing everything in memory is not feasible at scale, nor is it reflective of the biological, cognitive process of memory. Thus, I develop an iterative abstraction process that maps from concept intents to a vector of object counts (assuming a canonical ordering of objects O_t), stored sparsely. This function performs the STORE function of the anomaly reasoning framework. Initially, K_0 is empty, since no concepts have been formed. For the lattice formed at time t, $\mathfrak{B}_t$, the execution of STORE$(K_t, \mathfrak{B}_t)$ first aligns the current set of attribute-values $\mathcal{M}_t$ with the known ordering of all seen attribute-values. Then it converts each concept $c \in \mathfrak{B}_t$ into a condensed representation $\tilde{c} = (\vec{b}, l)$, a pairing of a bitstring and list of identifiers, where

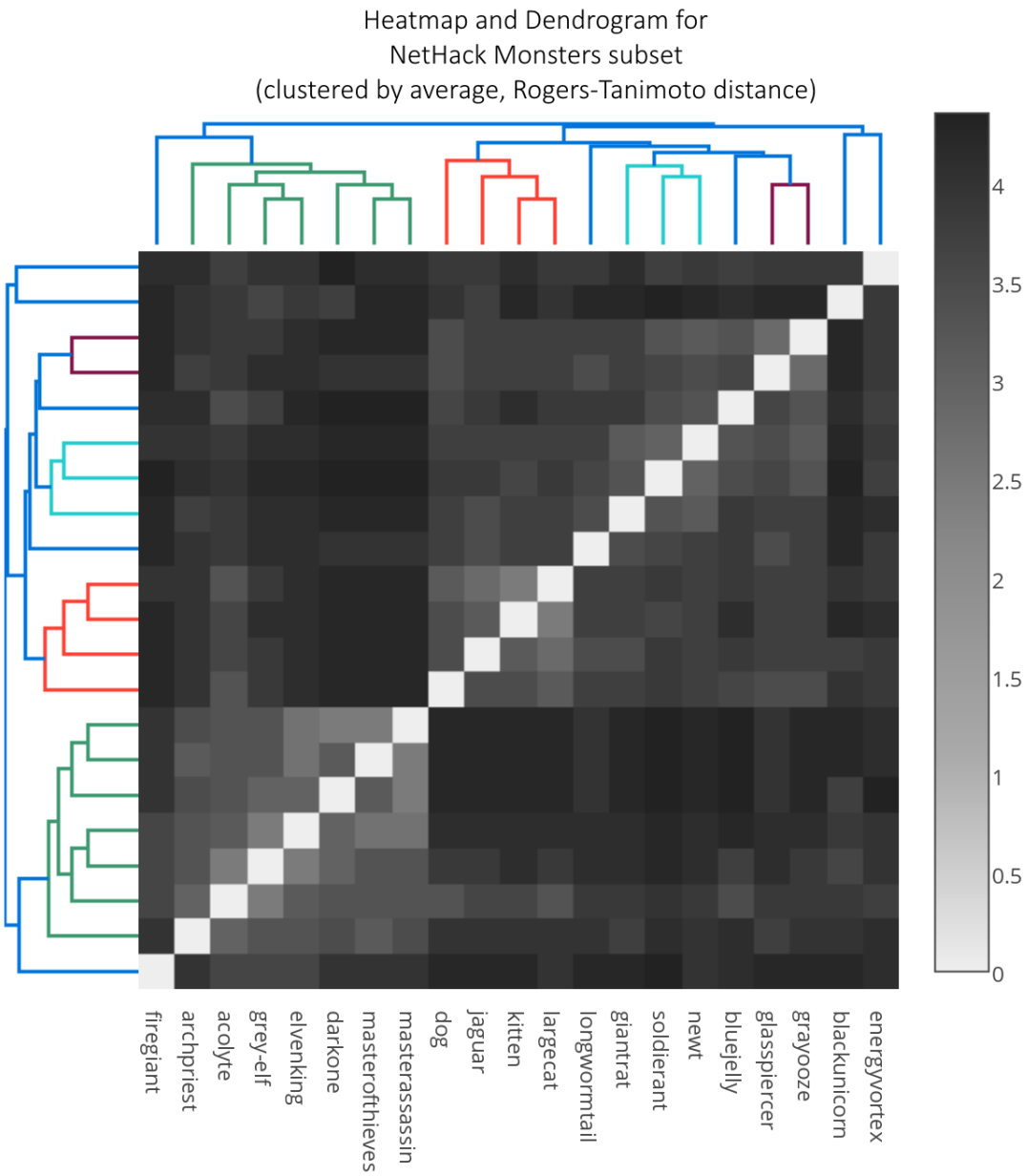


Figure 4.1: A heatmap and dendrogram for a subset from the NetHack data set. Clustering reveals semantically meaningful groupings of similar monsters and their relatedness to other subclusters.

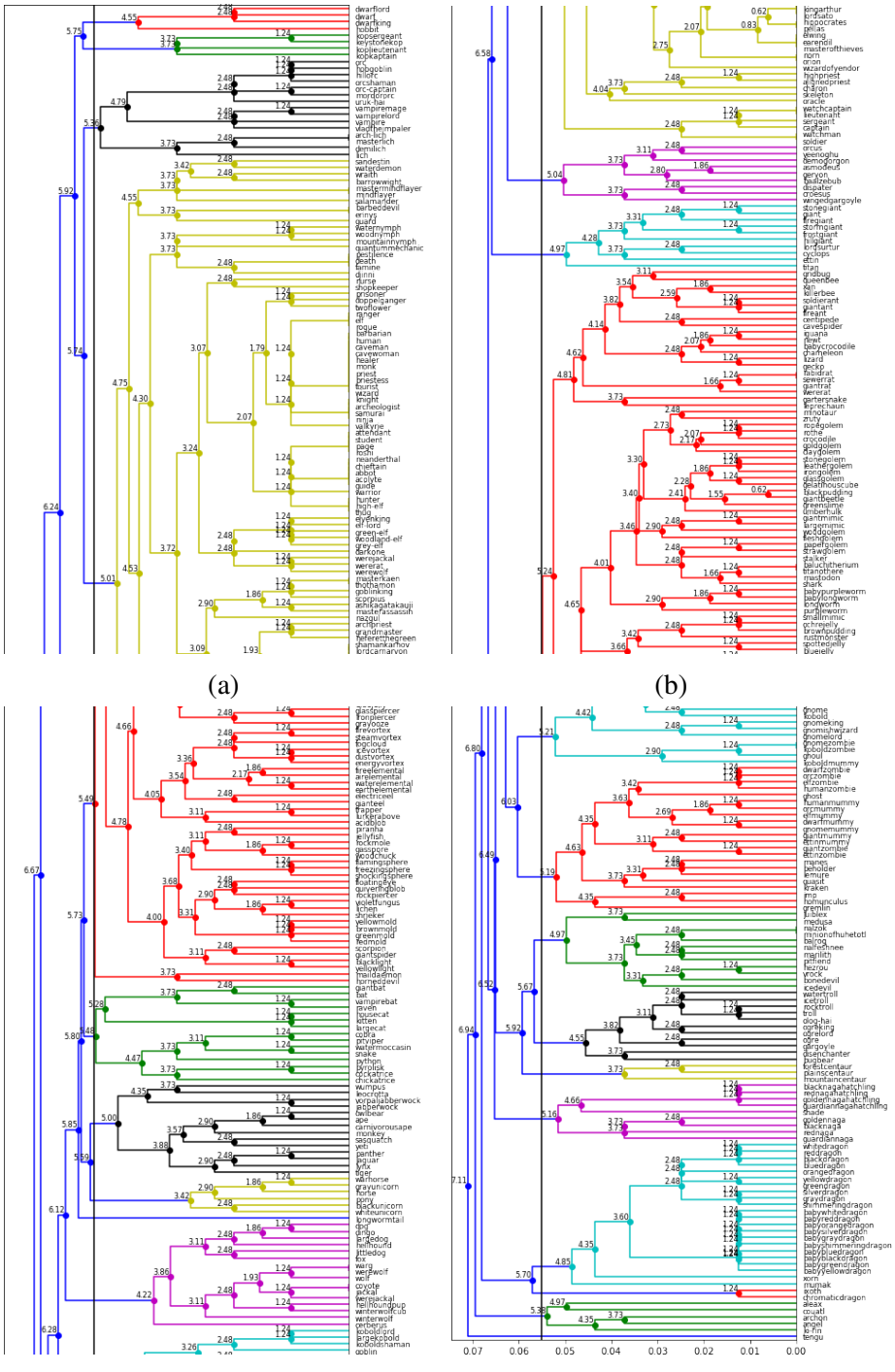


Figure 4.2: Dendrogram for the NetHack Monsters data set, clustered by average Hamming distance. The vertical axis drawn at distance 0.055 was chosen as the threshold, producing 22 clusters. Branch labels are the distance between subclusters scaled up ($d * 100$).

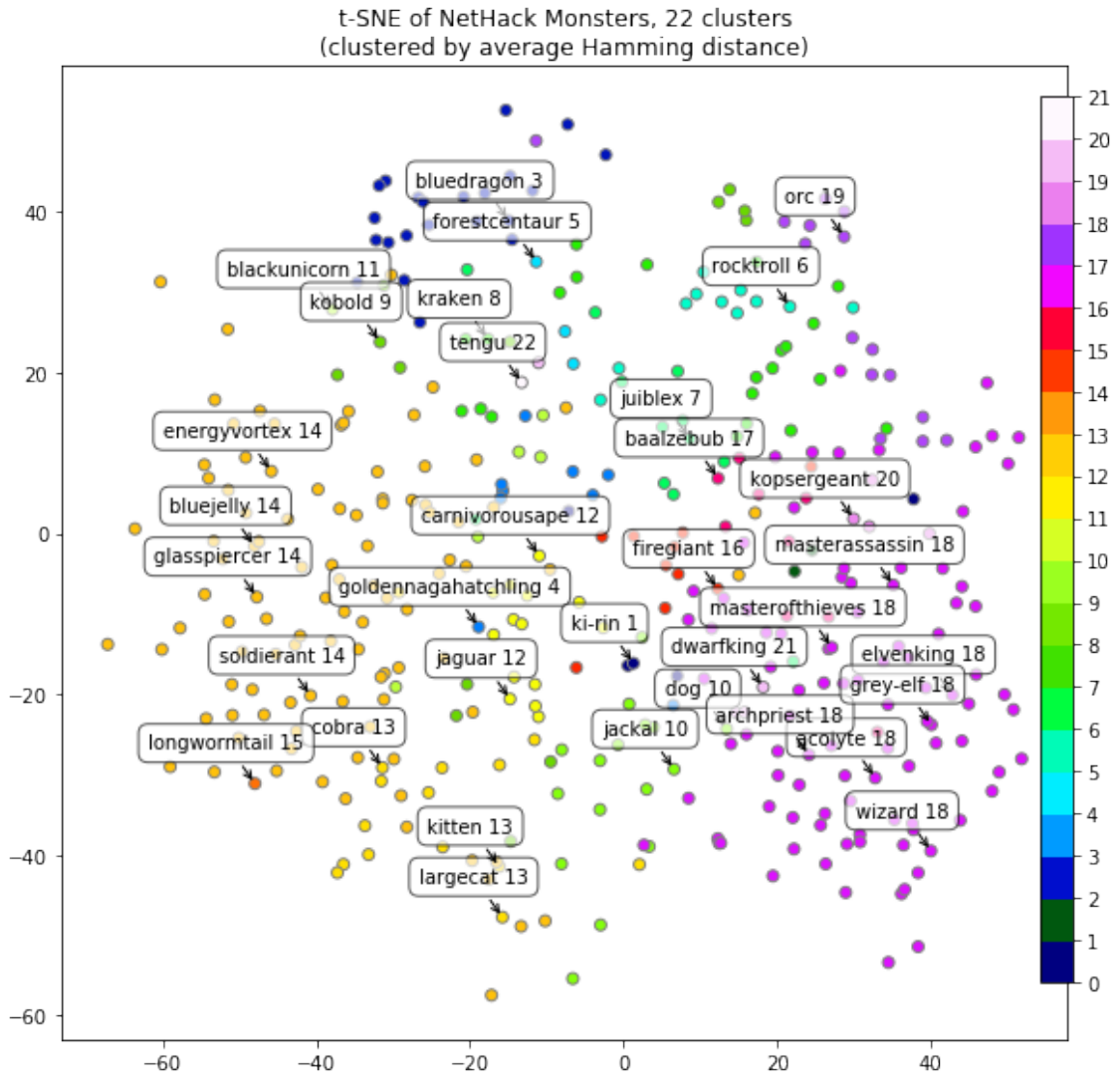


Figure 4.3: A t-distributed stochastic neighbor embedding (t-SNE) that visualizes the 22 clusters formed via clustering with a threshold of 0.055 on average Hamming distance. Some representative instances are labeled.

the former is from the binary vector of the concept's intent and the latter is simply the names or row labels of the objects in the extent. All such pairs are added to the knowledge base, $K_t \leftarrow K_{t-1} \cup \{\tilde{c} \mid c \in \mathfrak{B}_t\}$. This abstracted representation of the lattice can be stored efficiently, with each concept keyed on its intent's bitstring, which permits a bijective mapping to the identifier list. Whenever a known concept is observed ($\tilde{c}$ was already in K_{t-1}), its list l is simply appended to the one already stored in K_{t-1}. Equivalently for the NetHack Monster data set, a counter of each observed object identifier is incremented instead of using a list, which follows since an object's identifier is the name of its class. For example, having seen a "gnome" with the concept of "color:red" before and upon seeing one for the second time, the count of "gnome" stored in the bitstring corresponding to "color:red" would now be set to two.

The structure of K resulting from STORE is a partially ordered set of condensed concepts. It is not necessarily a lattice since there may exist a subset of two pairs that do not have a least upper bound or greatest lower bound (that is, they violate the join or meet conditions for a lattice). However, the lattice property is not needed for interpretation, only the partial ordering which is preserved through this procedure. Further, the condensed concepts in K_t can be totally ordered by their $\vec{b}$, which facilitates fast operations such as using a red-black tree for $O(\log(n))$ search, insertion, and deletion.

4.2.4 Analysis

In terms of assessing performance, my objective is to investigate the representative power of FCA. I describe and implement a simplified version of anomaly interpretation, hy-

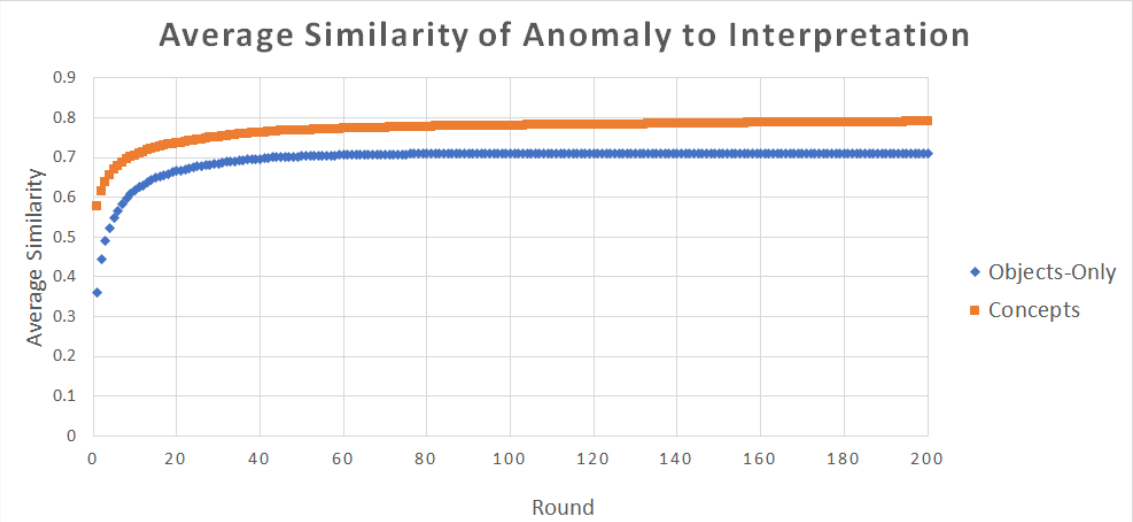


Figure 4.4: The average Rogers-Tanimoto similarity of anomalies to their interpreted (nearest neighbor) averaged across 25 trials. The data set was split into 80/20 partitions and evaluated each round. The similarity values are normalized with respect to the minimum similarity. Both the object-only and concept-based interpretations follow the same trend, but allowing concepts immediately increases and sustains the average similarity (the difference is statistically significant, $p < 0.05$).

pothesizing that it will be more accurate in some if not most cases to match anomalous objects to known concepts rather than previously stored cases (objects) alone. That is, I investigate performing nearest neighbor lookup by mapping objects into a conceptual space rather than finding similarity among known objects. The intention with this interpretation prototype is not to develop the most accurate nearest-neighbor method, but to create a diagnostic tool that demonstrates the utility of FCA for storing and condensing knowledge, mapping anomalous entities to analogous objects previously seen at various levels of abstraction.

My setup to analyze FCA-based interpretation follows a standard machine learning scheme, with a data set is split into training and testing, and an agent learns to map novel, unseen objects into the space of known, related data. The idea for iteratively abstracting over windows of monsters is to simulate what an agent playing NetHack might experi-

ence. That is, in any given state the agent would not see all possible monsters but only (at most) a small random subset comprising the formal context at that time. Using a train-test split of 80/20, an agent is given a subset of 15 monsters (drawn randomly with replacement) for 200 rounds each trial, for 25 trials. In each round t, a concept lattice is extracted from the context, abstracted, and stored in K_t. In evaluation, each test instance (an anomaly) is considered separately (i.e., not in a context) and against the nearest neighbor concepts according to various similarity metrics discussed in Section 4.1.4.2: Euclidean, Hamming, Jaccard, and Rogers-Tanimoto. To gauge the performance of concept-based interpretation with a simple baseline, I also compare it with an "objects-only" approach that followed the same process, but only stored objects and no concepts.

After training, the final $|K_{200}| \approx 700$ on average, meaning that slightly over half of the possible concepts from the entire NetHack Monsters data set were discovered after 200 rounds. Figure 4.4 plots the average Rogers-Tanimoto similarity, averaged across all 25 trials and normalized by the minimum similarity (i.e., the worst case interpretation). The plots follow a logarithmic trend, quickly reaching a threshold by around 60 rounds. The two lines are statistically significantly different (t-test at significance level of 0.05). Note that the concept-based interpretation immediately finds more accurate interpretations, sustains its improvement over the objects-only approach, and exhibits steps of increasing similarity (indicating more useful concepts are discovered periodically).

In general for concept-based interpretation, the majority of anomalies are interpreted as concepts that closely approximate the anomaly given the agent's knowledge. If the anomaly and its interpretation differ by one or two attributes, then they would have normalized Rogers-Tanimoto similarity of either 0.795 or 0.593, respectively. So, the re-

ported scores that exceed these similarity values indicate that the concept-based approach can recover interpretations that, on average, are more accurate. More precisely, the reconstruction and the original differ by strictly less than two bit positions (the interpretation is off by less than two attributes). This observation includes swapping the same attribute, for instance as with the anomaly "vampirebat" which is interpreted as a raven, identical on all attributes except it was assumed to squawk instead of squeak:

vampirebat [shape:BAT, color:BLACK, move:20;22, align:0;2, sound:SQEEK, size:SMALL]

interpreted as:

intent: [shape:BAT, color:BLACK, move:20;22, align:0;2, sound:SQAWK, size:SMALL]

unused: [sound:SQEEK], added [sound:SQAWK]

extent: [raven=29]

similarity: 0.976, normalized: 0.593

Two other examples that evince a close matching of concepts, where only an attribute-value was swapped for another, are the snake anomaly interpreted as a watermoccasin, and the cobra interpreted as a pitviper:

snake [shape:SNAKE, color:BROWN, move:15;16, align:0;2, sound:HISS, size:SMALL]

interpreted as:

intent: [shape:SNAKE, color:RED, move:15;16, align:0;2, sound:HISS, size:SMALL]

unused: [color:BROWN], added: [color:RED]

extent: [watermoccasin=31]

similarity: 0.976, normalized: 0.593

cobra [shape:SNAKE, color:BLUE, move:18;20, align:0;2, sound:HISS, size:MEDIUM]

interpreted as:

intent: [shape:SNAKE, color:BLUE, move:15;16, align:0;2, sound:HISS, size:MEDIUM]

unused: [move:18;20], added: [move:15;16]

extent: [pitviper=30]

similarity: 0.976, normalized: 0.593

A few anomalies are interpreted as an exact match, when the same combination of attributes have been observed before (e.g., coyote and jackal appear the same):

ranger [shape:HUMAN, color:DOMESTIC, move:12;13, align:-10;-2, sound:HUMAN, size:HUMAN]

interpreted as:

intent: [shape:HUMAN, color:DOMESTIC, move:12;13, align:-10;-2, sound:HUMAN, size:HUMAN]

unused: [], added: []

extent: [rogue=35, elf=36]

similarity: 1.000, normalized: 1.000

coyote [shape:DOG, color:BROWN, move:12;13, align:0;2, sound:BARK, size:SMALL]

interpreted as:

intent: [shape:DOG, color:BROWN, move:12;13, align:0;2, sound:BARK, size:SMALL]

unused: [], added: []

extent: [jackal=33]

similarity: 1.000, normalized: 1.000

Some other representative examples of successful interpretations include some that simply ignore attributes, when the anomaly best matches a more abstract concept:

panther [shape:FELINE, color:BLACK, move:15;16, align:0;2, sound:GROWL, size:LARGE]

interpreted as:

intent: [shape:FELINE, move:15;16, align:0;2, sound:GROWL]

unused: [color:BLACK, size:LARGE], added: []

extent: [lynx=2, jaguar=2]

similarity: 0.976, normalized: 0.593

---

vampire [shape:VAMPIRE, color:RED, move:12;13, align:-10;-2, sound:VAMPIRE, size:HUMAN]

interpreted as:

intent: [shape:VAMPIRE, align:-10;-2, sound:VAMPIRE, size:HUMAN]

unused: [color:RED, move:12;13], added: []

extent: [vladtheimpaler=1, vampirelord=1]

similarity: 0.976, normalized: 0.593

Only a couple of instances seem to defy logic or have poor similarity. One possible explanation is that this observation is an artifact of the data set, and that not every monster could group sensibly with others. Potentially there was simply no adequately similar concept ever observed during in a subset during training process. However, the dendrograms

previously discussed demonstrates that the following examples should have several candidate monsters that would serve as better nearest neighbors than the ones listed in the extent of the interpretation output. Still, even in cases where the interpretation finds low similarity, this information may prove useful to anomaly reasoning more generally. For instance, consider an agent playing NetHack encountered an anomaly such as the ones below. This agent could reason about uncertainty by leveraging poor similarity scores in its own position: such low values essentially inform the agent that it "knows it does not know" about the anomaly. The subsequent adaptation procedure could build on this explicit knowledge of uncertainty to guide exploration or evasion.

dwarf [shape:HUMANOID, color:RED, move:6;7, align:2;5, sound:HUMAN, size:HUMAN]

interpreted as:

intent: [color:RED, move:6;7]

unused: [shape:HUMANOID, align:2;5, sound:HUMANOID, size:HUMAN], added: []

extent: [pyrolisk=3, giant=3, hezrou=1, largekobold=1, dwarfzombie=3, pitfiend=5]

similarity: 0.952, normalized: 0.195

---

hobbit [shape:HUMANOID, color:GREEN, move:9;10, align:6;11, sound:HUMAN, size:SMALL]

interpreted as:

intent: [color:GREEN, align:6;11]

unused: [shape:HUMANOID, move:9;10, sound:HUMAN, size:SMALL], added []

extent: [couatl=3, greendragon=3, guardiannaga=2]

similarity: 0.952, normalized: 0.195

In some cases, even interpretations that yield low similarity can still have semantic validity. For instance, the "wingedgargoyle" anomaly has uncommon attributes, making it distant from most other monsters. Although it has low similarity to its interpreted concept, the others in the extent are in the same cluster as in Figure 4.2b, making it still relevant given the larger scope of the data set.

wingedgargoyle [shape:GREMLIN, color:LORD, move:15;16, align:-13;-10, sound:GRUNT, size:HUMAN]

interpreted as:

intent: [color:LORD, move:15;16]

unused: [shape:GREMLIN, align:-13;-10, sound:GRUNT, size:HUMAN], added []

extent: [croesus=1, demogorgon=1]

similarity: 0.952, normalized: 0.195

While this analysis is qualitative, it demonstrates that hierarchical concept-based interpretation is promising for attribute-rich domains. Extensions to supervised learning could focus on classification of anomalies (through concepts) by their cluster label or the unobserved "level" attribute from NetHack (corresponding to the relative power of the given monster). In subsequent chapters, I shift focus beyond mere classification to decision making, first in a contextual bandits framework, and then in a reinforcement learning paradigm.

Chapter 5: Concept-Aware Decision Making

In general human understanding, abstract concepts arise from our innate ability to recognize commonalities and patterns in the environment. Forming concepts aids our categorization of the objects we perceive in the world, enabling us to store and recall those things previously experienced. Beyond their use in analogical comparisons of the unknown to the known, concepts jumpstart our faculty for inductive reasoning. In other words, we use concepts in our practice of generalizing from few cases to many.

With concepts, we draw connections not just over the observational properties, or *attributes*, of objects, but over our *interactions* with objects. Consider billiards: just prior to our pool cue striking a ball, we anticipate the outcome, that the cue will halt and the ball will be propelled onward, arcing and ricocheting across the table. Even though we have perhaps never held this particular cue or struck this particular ball, we intuit the expected causal effects of this action, no matter how faulty induction may be [65]. Our concept of a pool cue interacts with our concept of a billiard ball to yield the changes in the world that we seek. From this vantage, we rely on conceptual abstractions of the world ubiquitously when deciding how to act. In this chapter, I tackle this problem, applying concept formation to decision making. Bridging from our previous discussion of concepts of taxonomic classification of anomalies, *concept-aware* methods for reasoning

intelligently about how concepts affect the results of our actions, and conversely how our interactions with the world bring about new concepts.

While *concept* is a vague, so-called "suitcase" term, I define it precisely based on the theory of formal concept analysis. My examination of formal concepts is extended from the previous chapter to consider them in a decision-making context, where anomaly interpretation and adaptation are handled together by an intelligent agent processing its state and taking actions to optimize discounted reward. In Section 5.1, I first introduce a novel form of state abstraction called *concept-aware feature extraction*, using FCA principles to create a new feature space that describes an agent's surroundings, from ground-level object-concepts up to increasingly more general clusters of objects' attributes. This approach is based upon my previously published work in Winder and desJardins (2018) [170]. In Section 5.2, I address the topic of contextual bandits, and demonstrate how conceptual features allow an agent to adapt more readily to novel and uncertain objects as they appear. In Section 5.3, my investigation shifts to temporal difference reinforcement learning, examining the beneficial effects to transfer learning that can be observed when concepts serve as the bases of linear value function approximation. I conclude with a summary of how concept formation helps transfer knowledge and behaviors learned in one context or task across related ones in unseen environments. Ultimately, concepts offer an interpretable and straightforward representation well-suited for agents to learn at varying, appropriate levels of generalization.

5.1 Concept-Aware Feature Extraction

Having established concept formation in Chapter 4 as a tool for handling novelty and subsuming the uncertain into the familiar, I aspire to incorporate this ability into intelligent agents that solve tasks. Moreover, my aim is to have concepts facilitate the transfer of knowledge across a universe of related tasks. Transfer in RL aims to have agents record and persist skills associated with features in their environment to better solve new challenges [154]. Concept formation permits knowledge learned from one task to be applied to a new problem by identifying the appropriate level of generalization, transferring behaviors between related tasks in unseen environments.

I introduce my novel technique, *concept-aware feature extraction* (CAFE), describing it first at a high level, and with greater detail in the following section. Using the scheme of concept formation for RL, I then consider its application to contextual bandits as well as reinforcement learning problems with factored state spaces (in which a state is comprised of a set of objects, each described in turn by a vector of attribute-values).

CAFE performs state abstraction by mapping a state to a vector of features. At its core, CAFE applies FCA to build a concept lattice from each state it encounters. The formal concepts, generated from state factors, form the basis of the derived features later used in function approximation. These concept-based features effectively serve as high-level descriptors or "hidden features" of state space.

By applying FCA, I leverage a central property to its theory: that it creates a partial ordering of factors. More precisely, the inherent super- and subset relationships of formal concepts means they delineate explicit tiers of increasingly abstract factor clusters

(the partial ordering) that recur across observed states. An agent's challenge, then, is to record experiences, such as value or expected reward, and learn behaviors over all concepts pulled from the corresponding lattices of observed states. In subsequent sections, I describe how agents learn how to act with concepts.

5.1.1 CAFE Approach

CAFE achieves state abstraction through a process of clustering, abstraction, and re-grounding. First, a concept-aware agent uses FCA to project its current state into an intermediate space of feature clusters (formal concepts). These concepts are grounded back onto the state, producing the final sparse set of features suitable for use in value function approximation. CAFE thus re-represents states in terms of the concepts that were extracted from them.

Definition 5.1 (Concept-Aware Feature Extraction). *The state abstraction function $\phi : \mathcal{S} \rightarrow \{0,1\}^k$ implements concept-aware feature extraction (CAFE) by following this procedure:*

1. *The input state s is converted into a formal context, $\rho(s) \rightarrow (\mathcal{O}, \mathcal{M}, I)$.*
2. *A concept lattice is obtained, $\Gamma(\mathcal{O}, \mathcal{M}, I) \rightarrow \mathfrak{B}$.*
3. *For each formal concept $(\{m\}^{\downarrow}, \{m\}) \in \mathfrak{B}$, an intermediate abstract concept is created: $c = (|\{m\}^{\downarrow}|, \{m\})$, the object count and attribute set, for $c \in \mathcal{C}$.*
4. *Concepts are grounded to s, yielding concept substates, $\Psi(s, \mathcal{C}) \rightarrow \mathcal{Z}_s$. Note that $\mathcal{Z}_s \subseteq \mathcal{Z}$, the set of all known substates, with $k = |\mathcal{Z}|$.*
5. *In the output vector ϕ, element $i = 1$ if concept substate $z_i \in \mathcal{Z}_s$, 0 otherwise.*

Thus, ϕ produces a sparse feature vector by composing functions ρ, an object recognizer; Γ, a concept-mining algorithm; and Ψ, a grounding procedure.

From Definition 5.1, I see that CAFE combines several functions to express the

abstract concepts that are present in a ground state. A concept-aware agent, thus, goes through concept formation and then concept grounding, localizing the abstract concepts to their current context.

Concept Formation

First, states must be represented in terms of a formal context, as the incidence I of objects $\mathcal{O}$ on attribute-values $\mathcal{M}$. For an OO-MDP state, the set of objects may serve immediately as the complete context, though I allow any general ρ mapping of states to be used. CAFE then computes a concept lattice from this context using Γ, which may be any standard concept-mining algorithm such as IN-CLOSE2 [7]. The output is a concept lattice $\mathfrak{B}$, a partial ordering of formal concepts of the form $(\{m\}^{\downarrow}, \{m\})$, paired sets of member objects and attribute-values, respectively. To further abstract away the specific objects, each formal concept is converted into an *abstract concept* pair, $c = (|\{m\}^{\downarrow}|, \{m\})$, retaining the attribute set but replacing the object set with a count. Using the object count achieves abstraction while still permitting us to differentiate among concepts that share the same $\{m\}$ but not the same number of objects (for example, the concept of two green squares versus that of three green squares). Note, I refer to the set of formal concepts extracted as the lattice $\mathfrak{B}$, separate from the set of abstract concepts $c \in \mathcal{C}$.

Concept Grounding

Next, each abstract concept is grounded to the state s. This process filters the objects in the agent's current surroundings by the high-level cluster of features identified via FCA.

I define a grounding procedure Ψ that produces a set of *concept substates*, the result of grounding abstract concepts to the original state. I regard z as a substate because it describes the pieces of the ground state that remain when filtered through the lens of its particular c. These are precisely the kind of "hypothetical states" discussed in Section 4.1.4.4. Formally, $\Psi(s, \mathcal{C}) \rightarrow \mathcal{Z}_s$. Individually, each $c \in \mathcal{C}$ is grounded with a subprocedure $\psi(s, c) \rightarrow z$, and their collection forms the set $\mathcal{Z}_s$. Additionally, I assume an arbitrary ordering of $z \in \mathcal{Z}$ based on the order in which they are encountered.

The ψ function constructs z by leveraging the factored nature of ground states. Specifically, psi duplicates s to get z, and then removes all factors corresponding to objects and attribute-values not present in c. An implementation question arises in the case that, as designers, we wish to preserve some factors from being abstracted away. Suppose we want to keep a feature from figuring into concept formation, or otherwise preserve a feature unique to an object that should persist through abstraction; for example, an identifier feature or label. We can straightforwardly prevent features like identifiers from inclusion in the formal context. However, upon grounding, knowledge of them would be lost without measures to retain them. Thus, in grounding the abstract concept back to the state, the identifiers would be preserved in the substates, remaining on any objects not fully abstracted away. Should no factors need to be preserved, I allow that grounding may be implicit, with the abstract concepts $\mathcal{C}$ serving directly as substates $\mathcal{Z}$.

For an example of grounding, consider a state that contains one red chair, one blue chair, and one red backpack (Figure 5.1). If this state is abstracted by the concept "red," the resulting concept state would consist of two red objects (subtracting all shapes, other colors, and objects not matching any attribute-value in the concept). Similarly, if that

same state is abstracted by the concept "chair," it would produce a concept state of solely two chair objects.

Featurization

Finally, featurization by ϕ produces a vector indicating the presence or absence of concept substates. As an ordering of $\mathcal{Z}$ is assumed, it follows that one then computes the vector resulting from $\phi(s)$ such that each element ϕ_i is equal either to 1 if $z_i \in \mathcal{Z}_s$ (0, otherwise), up to $k = |\mathcal{Z}|$ elements.

I highlight the fact that this vector is sparse, in that CAFE only needs to store information related to concepts already encountered (and not for all possible concepts). As more concepts are discovered, k increases. Yet, older outputs of ϕ need not change, except by appending 0 elements to their vectors, since, due to the properties of FCA, any new concepts by definition must never have existed in a previously seen state. Thus, concept substates may be reduced to the set of those known incrementally, as they are encountered. Therefore, one may say that CAFE is extensible, without the need to update or recompute any prior knowledge or representation, and elaborate further in Section 5.1.3.

In summation, CAFE effectively maps states into *abstract concept space*, the vector space spanned by the basis functions corresponding to the abstract concepts of all extracted lattices.

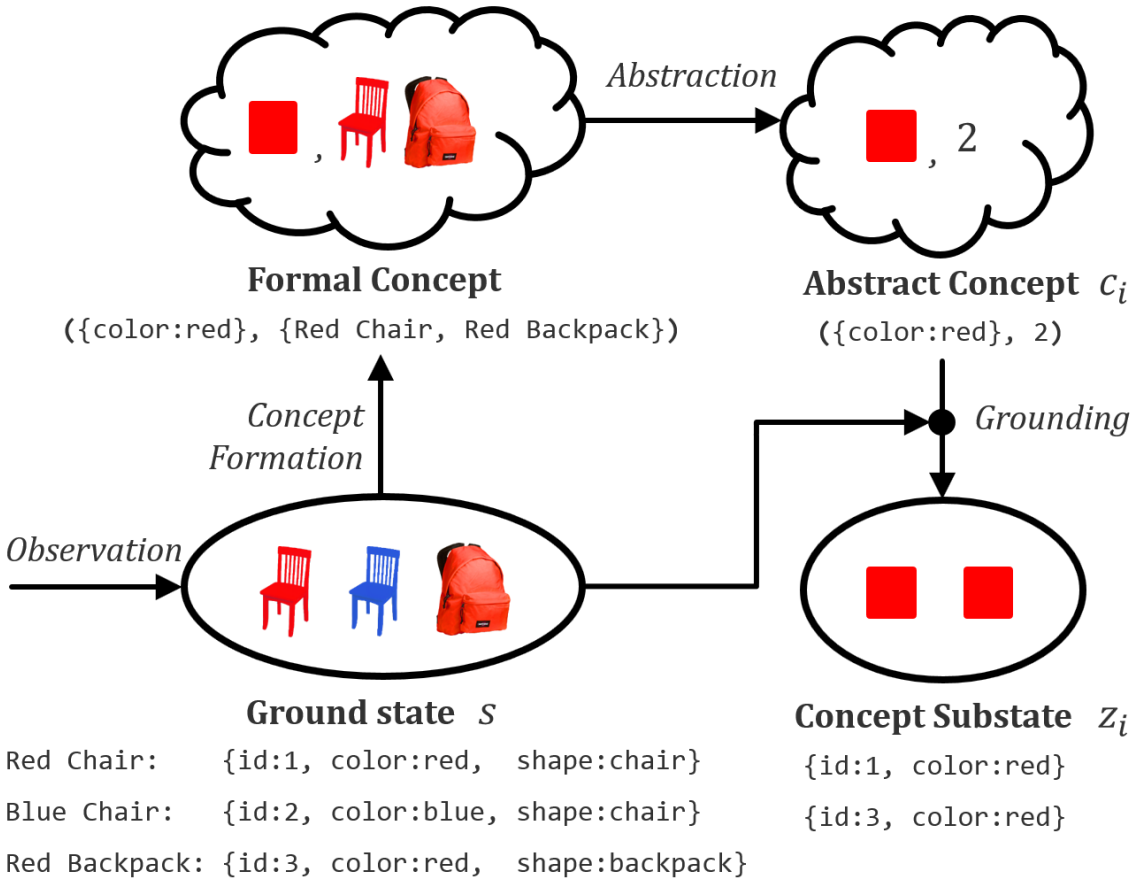


Figure 5.1: A diagram of CAFE for an example concept. Consider a toy example of three objects, each with a color and shape, as well as an attribute "id" that we, as designers, have marked as held back from state abstraction. Given the current state (observation), concepts are extracted using FCA (concept formation). Now regard just one example formal concept corresponding to the attribute-value "color:red", in the upper-left. Its intermediate abstract concept c_i is obtained with an object count of two (abstraction). Combining c_i with s produces a concept substate z_i (grounding), a filtering of the ground state based on the concept, while preserving any held-back attribute-values. Note that this process is repeated *for each concept* formed; together, the set of concepts substates are used as features in VFA.

5.1.2 CAFE Anomaly Reasoning

In terms of the anomaly-reasoning framework from Chapter 4, CAFE satisfies the goals of identification and interpretation. Referring to Definition 5.1, the function ρ handles RECOGNIZE. In my future experiments, ρ is implicit, because my assumption of an OO-MDP means states are already represented as object sets, and anomalies are any unknown (never previously seen) states or objects. CAFE also encompasses the interpretation functions of CONCEPTUALIZE, STORE, and ABSTRACT. Γ performs CONCEPTUALIZE. Steps 3 and 4 address ABSTRACT, first abstracting formal concepts to remove object-specific references, and then creating concept substates via the grounding procedure Φ. In general, novel substates appear in the presence of any new, anomalous objects or features, creating the uncertainty to which an agent must adapt. A concept-aware agent, then, achieves anomaly adaptation by using CAFE in conjunction with a suitable decision-making algorithm.

For STORE, I assume that the anomaly-reasoning knowledge base K retains an ordering of the k concept substates that have ever been seen, expanding as new ones are encountered. For use in linear VFA, CAFE's K would also include a weight vector θ, also of length k, adding another weight whenever a new concept substate is observed due an anomaly. Thus, the representational space of domain knowledge grows to accommodate novelty.

Considering the theoretical size of K, I note first that in any given MDP, there exists some range of all possible attribute-values that any object could take, $\mathcal{M}$. Hence, $|\mathcal{Z}| = 2^{|\mathcal{M}|}$, the theoretical extent of all knowable concept substates, as derived from

the powerset of all attribute-values. However, as previously discussed, CAFE employs sparsity and does not need to represent any concept substates not yet seen. Additionally, in practice, I find that the number of possible concept substates far exceeds the set of those explicitly known to an agent, $|K| << |\mathcal{Z}|$. Recall that this thesis is approaching decision making from the goals of transfer learning and generalization. Thus, the domains under consideration are those in which, from state to state, there exists considerable overlap in the objects and attribute-values that are present. Indeed, this assumption is crucial for generalization to be possible. Thus, I would not recommend CAFE for pathological domains where states share nothing in common (such domains being ill-suited for transfer learning, violating a standard assumption of relatedness among tasks). Again, I observe that even in the complex domains later discussed, the actual number of concepts seen and stored is orders of magnitude smaller than the theoretical extent of concept space.

5.1.3 CAFE Properties

I motivate CAFE as an automatic featurization technique following from FCA theory, where the set of available symbols expands naturally and hierarchically upon the encountering of novel, anomalous objects. My primary focus, then, is CAFE's use in contextual bandit and reinforcement learning problems.

Extensibility

As opposed to many other featurization techniques as discussed in Section 2.1.2 (e.g., tile-coding, radial basis functions, Kanerva encoding, deep convolutional neural networks),

CAFE can yield as many novel features as are necessary to describe a given state, extending its representational space as new features are encountered. I observe this property of CAFE is due to the inherently extensible nature of FCA, such that new formal concepts may be incorporated incrementally with their hierarchical (sub- and super-concept) relationships immediately apparent. This extensibility will be seen in the following sections to facilitate the transfer of learned behaviors effectively, where other approaches would falter or fail.

Beyond assuming the ability to define a ρ function that represents states as a formal context, CAFE does not need to know anything else about the structure or nature of state space. Crucially, other existing featurization methods are not designed to handle the case of an expanding set of features, with each making assumptions on the scope and properties of their input states. For example, to properly compute the tiles for CMAC, the precise number of state factors possible in any single state must be known beforehand. For neural networks, the input (often, a matrix unrolled to a vector) must be held constant, like a window or fixed sensor view of the world, such that "larger" states or windows cannot be represented without rescaling or similar transformations. For radial basis functions and similar approaches, the range of all possible object-attribute values must be known beforehand, along with the precise number of classes of objects. Thus, existing techniques assume that their featurization is *a priori* sufficiently complete to represent all future tasks.

Benefits of FCA

I purposefully rely on concept substates as the vehicle for featurization, as opposed to the more abstract formal concepts. The reason is that, in grounding concepts to the original state, it is possible to preserve and differentiate the objects inside the state. For example, suppose there are ground states s and s' such that s has two blue objects and s' has four blue objects. Now assume that conceptualizing the states creates a formal concept in both cases with an *intent* of "{color:blue}." Note that the resulting concepts differ on their *extents*: the one from s would have two objects, while s' would have four. Without differentiation among object support, a state abstraction would lack the granularity to capture this difference, as the "blue" concepts produced s and s' would both map to the same abstract intent. For CAFE to preserve the difference, grounding creates more expressive features based such that $\psi(s, c_{\text{blue}}) \neq \psi(s', c_{\text{blue}})$.

Overall, the FCA-driven featurization is advantageous, because it gives an automated process of finding concepts, which are portable, semantically relevant groupings of objects and attributes. Concepts also permit implication inference, clustering, and measuring various types of similarity, leading to greater model interpretability. Future explorations of CAFE not addressed in this work include using the hierarchical nature of concepts directly into the featurization and function approximation computation.

5.2 Contextual Bandits with Concepts

As a first inquiry into the use of concepts for sequential decision making, I leverage the formulation of such problems as contextual bandits (Section 2.2.1). For contextual

bandits, the arms comprise the action set, and the context of features describing the arms constitutes the state (along with any side information).

5.2.1 Object-Oriented Contextual Bandits

I begin my discussion by introducing a new, intuitive way to represent contextual bandits problems: object-oriented CB, a straightforward way of reframing of bandits from a perspective inspired by OO-MDPs. In this approach, a CB context is represented by an OO-MDP state. Objects now express either the features of an arm or of side information. The set of available actions are then parameterized over the objects in the state.

To highlight the representational difference, consider a simple problem domain with k slot machines, each described by m attributes. The normal contextual bandits setup would have k arms, and hence k actions that represent pulling the respective arm, with a context vector of $k \times m$ features. The OO-MDP framing instead casts each context a state with k slot machine objects. The OO-MDP action set consists of one "pull" action, parameterized over objects of the slot machine class. Though subtle, this difference between the standard CB and OO-MDP version is crucial. While the number of features remains the same in both cases, considering contexts to be object-oriented permits a straightforward and principled way of defining and extending problems. That is, any new side information may be incorporated as an additional object.

Consider the following variation of the slot machine scenario. Suppose there are both positive and negative payouts from the machines. Further suppose a new type of side information is added in the form of a beacon. When lit, the beacon indicates that

the payouts from the slot machines will be inverted. With the object-oriented version of the CB problem, the beacon can be added as its own object to the state directly, with no additional manipulation needed (since the action set was already parameterized over slot machines, and not beacons). Inclusion of another feature in the traditional contextual bandits may be handled trivially by appending it to the context vector, but expressing how the action set must be changed is trickier, especially if one eventually wants to have some side information affect only certain arms. Overall, an object-oriented approach to CB problems posits that the context may be, quite simply, a set of objects, with the arms being an action set parameterized over a subset of those objects.

5.2.2 Concept Features & Bandits

The contextual bandits with concepts (CBC) paradigm encompasses any approach using CAFE to create a lattice of concepts from a given context in a bandits setup. I introduce CAHL, a CBC algorithm, and describe precisely how concept formation aids decision making. I analyze the performance of CAHL relative to standard techniques based on LinUCB (Section 2.2.1) for a set of bandits domains.

5.2.2.1 Hybrid Linear Models

Since concepts are derived from all objects, and rely on the co-occurrence of those objects' attribute-values, I elect to employ *hybrid linear models* in my algorithms. I refer to LinUCB with hybrid linear models (Algorithm 2 in [90]) as HL, as opposed to LinUCB with disjoint linear models, which I call L. HL maintains individual models for each arm,

as in standard LinUCB, but also possesses a hybrid *joint model* that operates on a separate feature set potentially derived from all arms. Recall that L relies on independent models for arms. HL makes these models hybrid, and dependent on the context, with features derived from all arms. The set of shared features may be supplied arbitrarily, such as being computed by the outer product of all other features.

Formally, I distinguish the disjoint featurization of used by the arms, ϕ_a, from the ϕ_j featurization of the context shared via the joint model. Therefore, the hybrid approach must be extended beyond the standard, disjoint action-value models of LinUCB (as in Equation 2.10), with the summation of estimates from the joint and given action models:

$$\hat{Q}(s_t, a_t) = \mathbb{E}[r_t \mid s_t, a_t] = \theta_{a_t}^{\mathsf{T}} \phi_{a_t}(s_t) + \beta_t^{\mathsf{T}} \phi_j(s_t), \tag{5.1}$$

where β_t are the weights of the joint model as learned up to time t. Thus, I call HL the LinUCB algorithm with action model specified by Equation 5.1. As HL makes decisions, both the model for the arm chosen and the joint model are updated. Computation of ridge regression and confidence intervals with hybrid models requires some revision to factor in terms for the joint model and its parameters, though all the properties of UCB hold; for a more extensive discussion and HL's pseudocode, I refer the reader to Li et al. [90].

5.2.2.2 CAHL

The hybrid linear models framework facilitates a separation of features specific to arms, ϕ_a, from those generally applicable to the context, ϕ_j. In applying the framework of CBC to HL, I define an algorithm I call concept-aware hybrid LinUCB, or CAHL. Concretely

for CAHL, CAFE serves as hybrid LinUCB's ϕ_j such that the lattice extracted from the context and its derived features are shared among all arms.

CAHL considers a model of each arm separately (given its own features), but also learns the action-value of concepts as shared across the set of possible arms. The goal, thus, is for the shared concepts to provide a kernel of knowledge about the kinds of arms that might appear, constructed incrementally, such that CAHL may better interpret and adapt to a new anomalous arm. In this sense, CBC addresses the well-known problem of *exploration vs. exploitation*. By more gracefully characterizing anomalies the arise, concepts help bias one's interpretation of them to reduce the number of harmful rounds of exploration that would otherwise be needed.

5.2.3 CBC Methodology

I outline the following approaches to contextual bandits:

- L : standard LinUCB, such that ϕ_a features are the attributes of the given arm plus any side information, no ϕ_j,
- HL : standard hybrid LinUCB, such that ϕ_a is as with L, and ϕ_j are the features of all arms and side information,
- CAHL: variant of HL, such that ϕ_a is as with L, and ϕ_j are the concept features from computing CAFE on all arms and side information.

The disjoint arm models of L all apply the same ϕ_a featurization technique. For HL, the shared features ϕ_j are used only in the joint model. I follow the methodology of Li et

al. [90] and let ϕ_j be the outer product of all arm features and side information. CAHL builds directly upon HL by applying CAFE for ϕ_j. In essence, CAHL treats the context of contextual bandits as it would an OO-MDP state or formal context in FCA. For each method, I add a bias feature to all arm and joint models.

Together, I apply these algorithms to a variety of domains to investigate the empirical value of concepts and the ability of a CBC approach to handle novelty in the form of anomalous arms. A summary describing the set of bandits domains I consider follows:

- Synthetic: pick the best of n arms, where an arm's reward is the dot product of its m latent attributes and a context of m Gaussians [130],
- Mushroom: pick the one edible mushroom from a set of n mushrooms, where the others are poisonous [130],
- NetHack Monster level alignment: match the player's observed level to one of n monsters' unobserved levels.

Synthetic

For the Synthetic CB problem, the context contains m Gaussian objects and n arms. Each Gaussian, g, possesses one real-valued α attribute that varies stochastically, with its value at state s_t sampled according to $g.\alpha_t \sim \mathcal{N}(0.0, 1.0)$. Hence, a context at time t is defined as $G_t = \langle g_1.\alpha_t, \ldots g_m.\alpha_t \rangle$. An arm, a, possesses m real-valued β attributes that are hidden from the agent and held constant in every state over the duration of a trial. The value of an arm's β attributes are initially generated uniformly at random, $a.\beta_i \sim \mathcal{U}(0.0, 1.0)$. Further, each arm has two additional latent attributes, $a.\mu$ and $a.\sigma$, used to add Gaussian

noise to the reward the arm generates. Upon pulling arm a, reward is computed as the dot product of the arm's m unobserved attributes and the known context of the m Gaussian attributes, plus noise: $R(s_t, a) = a.\beta \cdot G_t + \mathcal{N}(a.\mu, a.\sigma^2)$. Arms, therefore, can be approximated well by linear models, so I include this domain as a baseline where the standard algorithm L is perfectly suited.

The Gaussian objects comprise side information (not intrinsic to the arms) and so are supplied to the model of each arm in L, HL, and CAHL. For the experiments I present, the setup follows Riquelme et al. [130], such that $m = 10$, $n = 8$, $a.\mu = 0$ for all arms, and each arm is given a different value of σ, from 0.0 to 0.7 in increments of 0.1.

For CAHL, because the domain contains continuous features, I apply a discretization technique based on rounding: before execution, a user some specifies integer d, the decimal places by which to round. Then, during featurization, a separate pseudo-feature is generated for each decimal place up to the one specified. As an example, if a user selects $d = 4$, then for a Gaussian with attribute $g.\alpha = 0.12395$, the discretization process (prior to concept-formation) would generate each of the following separate attribute-values: $g.\alpha$:0.1240, $g.\alpha$:0.124, $g.\alpha$:0.12, and $g.\alpha$:0.1.

Mushroom

The Mushroom data set [137], from the UCI machine learning repository [33], is a well-studied classification problem that has been cast as a bandits problem in recent literature [21, 130]. Specifically, in a typical CB formulation, a context consists of a set of five mushrooms (the arms), such that one is edible and the other four are poisonous. The

one action "eat" is parameterized over each potential mushroom. The goal is to select the edible mushroom and receive a guaranteed reward of 1.0. Should the agent select a poisonous mushroom instead, it would receive rewards of either 1.0 or -7.0 with equal likelihood. Each mushroom contains 22 observable attributes, crucially with one additional attribute unobserved (the toxicity label, either edible or poisonous). The challenge of the Mushroom domain for CB extends beyond classification by including the poisonous mushrooms as confounders that offer considerable noise, since they appear safe half of the time they are selected. Moreover, as with all bandits problems, only the reward of the selected mushroom can be observed; the counterfactual reward of those not picked remains mysterious. Unlike the Synthetic CB problem, there is no side information, only arm-specific features.

Building a concept lattice on the full Mushroom data set yields around 220,000 formal concepts, depending on the preprocessing of the data [8]. For these experiments, I remove the mushrooms with missing attributes, leaving a set of 8124 potential mushrooms, making it extremely unlikely an agent would ever see the same context twice. Additionally, to keep the total number of concepts used in CAHL models reasonable, I limit the minimum support of concepts extracted by CAFE to 3.

NetHack Monster

In the NetHack Monster level alignment CB problem, the agent is presented with a context consisting of n monsters (the arms) and a player object (side information). The player has an observed level (an integer representing experience and relative ability), and each

monster has a set of observable features as well as an unobserved level. The goal is for the agent to select the monster that has the closest level to its own; the goal reward is 1.0 if the levels match, otherwise reward is -0.1 multiplied by the distance between their levels. To generate the arms, one action type "attack" is parameterized over each monster object in a given context. An agent must choose the monster most analogous to itself in terms of level. I motivate this goal as a proxy simulating the choice an agent would routinely face while playing NetHack: among the monsters in its current context, an agent must decide which is most worthwhile to attack (with a level most similar to its own experience), such that weaker or much stronger monsters have diminishing returns. The decision posed by this problem is similar to classification, but requires the algorithm to distinguish the best choice among several options, except with additional information of a label to match (the agent's level) given beforehand. Unlike both the Synthetic and Mushroom CB problems, this domain contains both side information and arm-specific features.

To create the data set for level alignment, I limit the monsters to those between levels 0 and 10, obtaining 277 monsters. For each episode of CB, there are $n = 5$ monsters available, with one guaranteed to have an unobserved level matching that of the player.

5.2.4 CBC Results

I examine the utility of a CBC approach in the Synthetic, Mushroom, and NetHack Monster problems. Result plots include 95% confidence regions.

5.2.4.1 Synthetic Results

The results of L, HL, and CAHL in the Synthetic domain can be seen in Figure 5.2. The algorithm that achieves the best regret (reflected here as the higher reward curve) is L, the default LinUCB formulation. These results are as anticipated, since the arms of this synthetic problem can be nearly perfectly modeled by the linear assumptions of L. Because HL and CAHL compute a joint model, they require more exploration simply to learn over the shared features. However, the hybrid linear models simply add a burden to the sample complexity that is not offset by their value. More precisely, the rewards produced by the arms are linearly independent (depending only on the context of side information), so learning disjoint models of each arm separately is sufficient to solve this CB problem. Likewise, the regret of CAHL is worse than HL (it obtains less reward in Figure 5.2) because it is learning a joint model over even more features. Note that I include a Random agent here to show the benefit of learning; I omit Random from other experiments since the same trend is observed.

Where the asymptotic trends for both L and HL are roughly parallel, CAHL suffers a slightly more linear trajectory. I account for this observation in remarking that the non-linear expansion of concepts (through discretizing the Gaussian objects' attribute-values) makes it more difficult to construct reliable confidence regions in feature space. Moreover, learning concepts (or any shared features) does not add information to help solve the domain. Specifically, the reward function for an arm a depends on the linear combination of the Gaussian objects and it's latent $a.\beta$ attributes. Forming concepts on the Gaussian objects provides little to no predictive value, because the way they affect reward

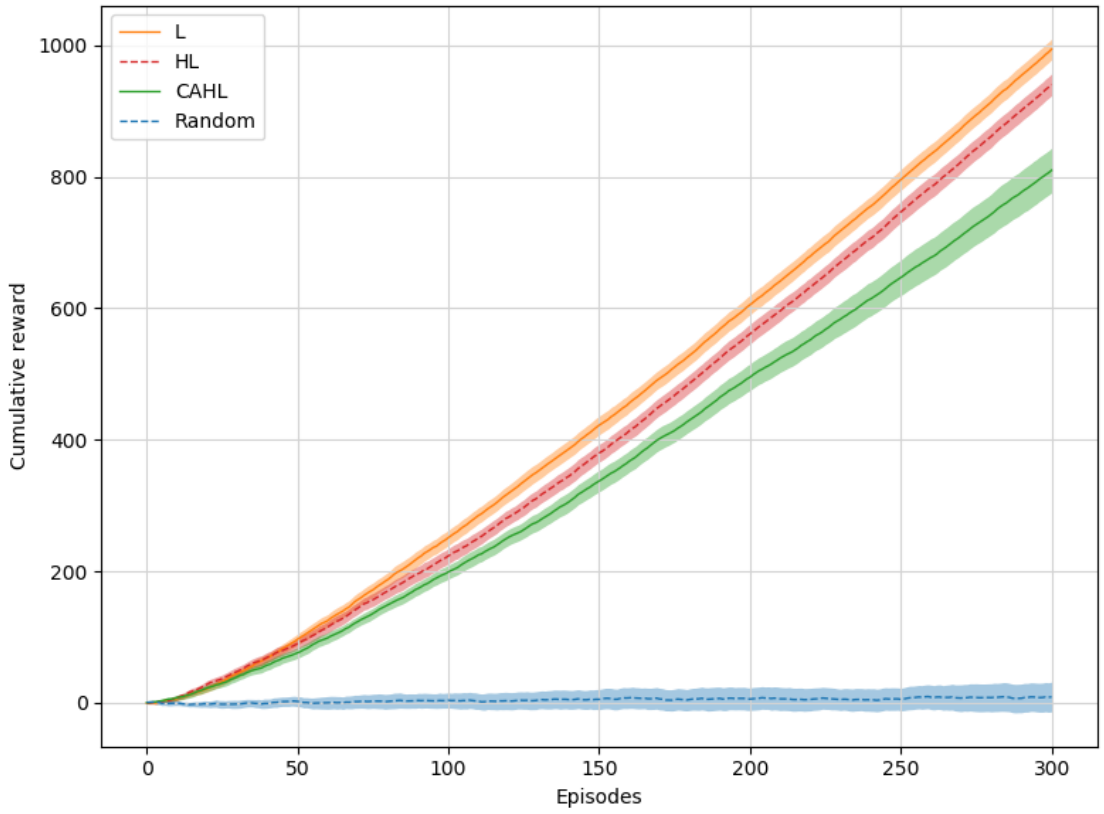


Figure 5.2: Results of LinUCB methods on the Synthetic CB problem, 30 trials.

is arbitrarily dissimilar across arms: the underlying β values are uncorrelated. Thus, since there is no extrinsic pattern or informative distribution underlying the generation of the β values of arms, no arm is like another. The Synthetic CB problem thereby eschews one of the key assumptions for anomaly reasoning and concept formation, that meaningful categories of objects exist in the environment, and can ultimately be seen as clusters in a vector space representation of the world. This domain plays an important role as a benchmark, demonstrating that even when the CBC assumptions are violated and applied to a continuous context where linear models are sufficient, CAHL can still learn approximate models, just under the caveat that feature complexity is traded for asymptotic performance.

5.2.4.2 Mushroom Results

Results on the Mushroom CB problem are reported in Figure 5.3. As opposed to the Synthetic problem where arms had no attributes and side information was passed along to each, here arms have many attributes and no side information. Because learning to estimate the expected reward of a mushroom is independent of other arms in the context, the disjoint models of L perform fairly well. By around 400 episodes, L attains a flatter reward curve and begins to readily pick out the edible mushroom while avoiding poisonous ones.

Conversely, HL struggles learn decent models in the 500 episodes plotted, attaining consistently worse regret. This observation may be explained in part due to HL's increased sample complexity over L, as in the Synthetic CB problem. HL requires more time to collect samples to learn the large number of weights for its shared features.

More weights are not inherently an impediment to learning, as is reflected by the results of CAHL. Though CAHL has a sizeable joint model and set of shared features, as HL does, it manages achieve a curve similar to and exceeding L, circumventing the apparent burden of sample complexity. Even in the earliest episodes, CAHL suffers less negative reward and begins to pull ahead of HL. This trend highlights how CAFE, though making minimal assumptions about the structure of data, can generate features that enhance learning in the face of uncertainty.

With each new context, CAHL must consider mushrooms it has never encountered before, thus generating one or more new concepts from these anomalies. However, these anomalies will yield concepts that CAHL *has* seen before, which allow it to weigh its de-

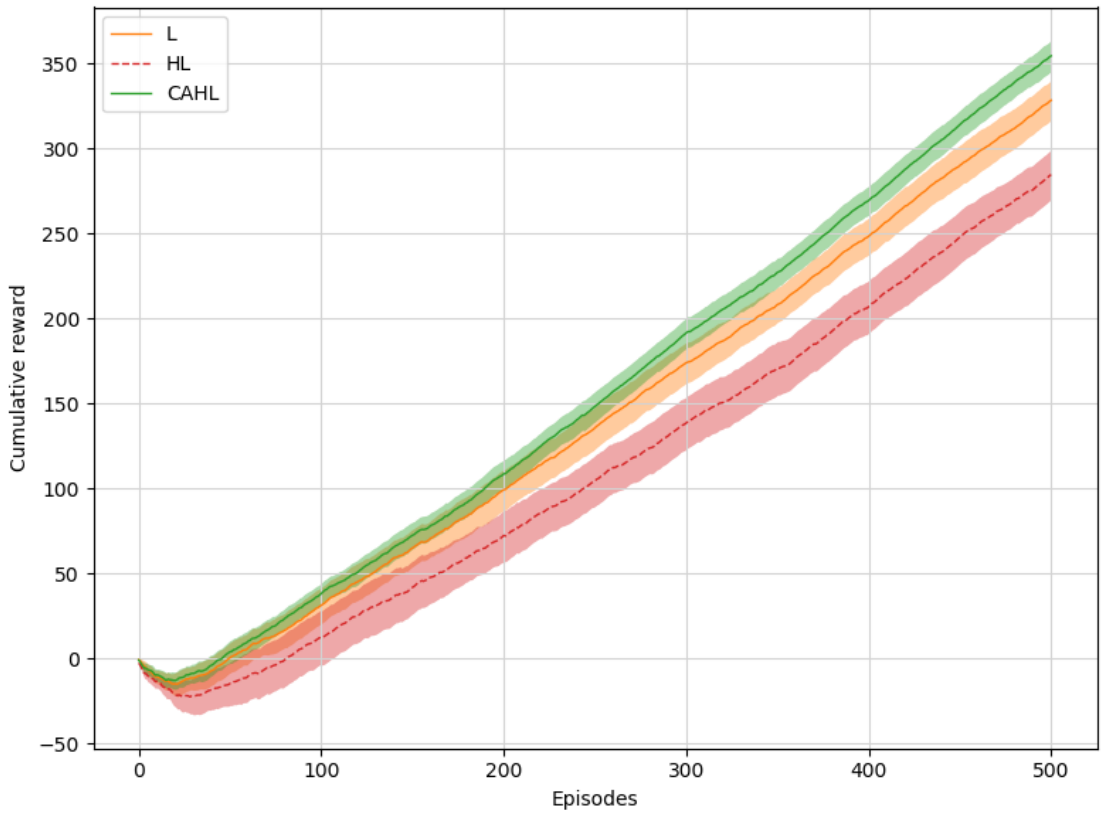


Figure 5.3: Results of LinUCB methods on the Mushroom CB problem, 30 trials.

cisions in terms of the anomaly's abstract similarity to what it knows. Recall that FCA allows us to find a natural hierarchy of objects, grouped and ordered by their attribute-vales, abstractly. Inherent to the use of CAFE, then, is the assumption that one can interpret and adapt to anomalies by their membership in these formal concepts. Here, that property suffices to account for the difference between CAHL and HL: CAFE's abstract features allow a more immediate and implicit mapping of novel objects to those that are similar, beyond what raw features allow. Thus, in applying CAHL to the "real-world" data of the Mushroom domain, it can be confirmed that CAFE is both reasonable and beneficial in a decision-making context. In Section 5.2.5 I further discuss how concept formation as a featurization technique offers worthwhile benefits over simpler models like L.

5.2.4.3 NetHack Monster Results

I present the results on the NetHack Monster level alignment CB problem in Figure 5.4. Due to the diversity and number of arms, as well as the presence of side information, learning takes much longer in this domain. The challenges of the previous tasks are intensified: not only must the agent select the correct arm given unobserved labels, it must condition this decision based on its own (observed) level. In a sense, my experiments have progressed from examining anomaly interpretation as a basic, naïve form of classification, through a kind of contextual classification as with the Mushroom domain, to the complex alignment problem faced here.

In Figure 5.4, CAHL consistently exceeds over L and HL, both in terms of raw cumulative reward and asymptotic trend. L and HL achieve roughly the same performance as each other. The stability of CAHL, as represented by the confidence region in Figure 5.4, also holds over the episodes. In contrast, for both L and HL, their variance in performance across trials increases as they go further out.

The learning curve of CAHL manifests a property of CAFE: early investment in exploration of (conceptual) feature space pays dividends in later performance. Until around episode 10000, CAHL achieves a reward that is about the same or worse as the other methods; beyond that, it excels. Initially, in the first hundreds of episodes, CAHL dips into a trough of negative reward. Upon examination of trial rollouts, CAHL consistently picks monsters that are not only new to it, but especially if they have unknown (and thus, dissimilar) concepts to what it has seen before. This application of UCB, then, means that CAHL pays a reward cost up front to learn about novel features. However, by approxi-

mately episode 5000, CAHL acquires good enough joint and arm models. From then on, it is on a steeper reward trajectory that ultimately surmounts L and HL, overcoming the trough of its initial negative performance. CAHL thus pays more to learn about concepts, but then uses this knowledge to handle new contexts more effectively.

From these collective observations it may be gathered that CAFE enables CAHL to adapt more readily. CAHL leverages the benefits of upper confidence exploration, while introducing CAFE's inherent transferability such that, as concepts are learned, that knowledge compounds its ability to interpret and adapt to anomalies. The problem of facing a novel monster is not just understanding that anomaly alone, but how it interacts with the agent's level attribute, relative to the other (potentially anomalous) monsters. CAHL's hybrid kernel of concept features means that anomalous objects are always interpreted in terms of their membership in multiple abstract clusters. This process translates into more informed, and therefore steadier and safer, decisions.

5.2.5 CBC Discussion & Analysis

My goal with these experiments is to gain insight into the practical benefits afforded by CAFE in a contextual bandits context.

5.2.5.1 Automatic Feature Clustering

As a feature generation technique, CAFE makes a minimal assumption about the nature of data by applying the theory of FCA. The result is a process of extracting meaningful, abstract features in a principled and automatic manner.

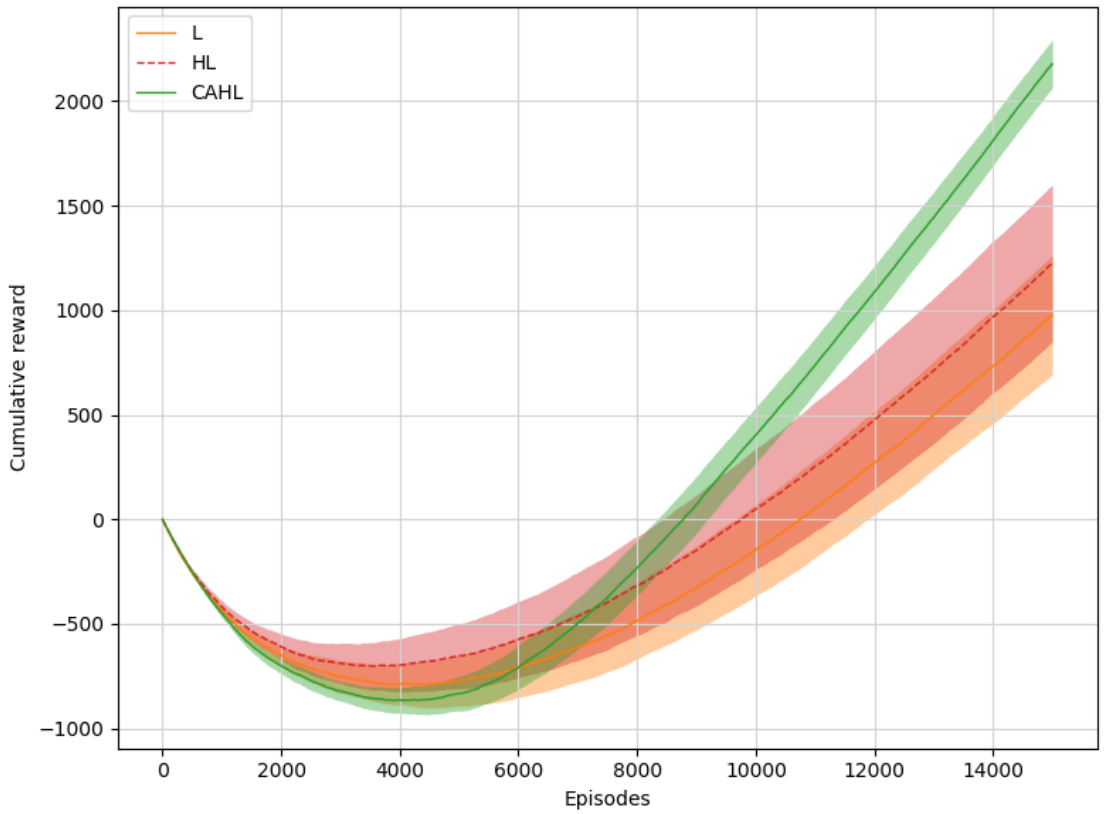


Figure 5.4: Results of LinUCB methods on the NetHack Monster level alignment CB problem, 30 trials.

L and HL consider the literal features of objects in the context. In contrast, CAHL incorporates new clusters of features on the fly, rather than enforce that new data adhere to procrustean constraints. For example, prior work has made the case for dimensionality reduction as a pre-processing step. By converting all input data (arm features) into a few clusters (e.g., six), contexts would essentially be encoded as low-dimensional embeddings [90]. However, this practice assumes up-front data collection, the application of offline cojoint analysis, and much expert engineering to design. My results reflect how CAFE achieves feature clustering automatically, while attaining improved performance using an *expanded*, rather than collapsed, set of features. Specifically, as opposed to the small, dense embeddings of previous work, CAHL operates over sparse feature vectors on the order of hundreds to thousands of entries.

While the CBC approach requires no prior expert knowledge or intervention, I acknowledge that one hyper-parameter does remain: the minimum object support for concept formation. I suggest this setting may be configured by random search, intuition, or left simply at zero. This hyper-parameter does bring a benefit: it permits a tuneable level of abstraction, such that a designer can inspect how agents perform on increasingly restricted clusters of features.

5.2.5.2 Concept Meta-Graph Visualization

To visualize feature clusters, I develop software[1] that plots the partial ordering of all concept substate features extracted over the course of a trial, and their associated learned weights. I refer to this structure as the *concept meta-graph*.

[1] I use D3.js, https://d3js.org/.

Definition 5.2 (Concept Meta-Graph). *The concept meta-graph $\langle \mathcal{Z}, \leq \rangle$ is a partially ordered set of concept substates $\mathcal{Z}$ and the partial ordering relation $\leq$ on the intents of concept substates. Let $z.o \in \mathbb{N}$ be the concept's object support, and $z.m \subset \mathcal{M}$ represent its intent (a set of attribute-values). For $z_i, z_j \in \mathcal{Z}$, $z_i \leq z_j \Rightarrow z_i.m \subseteq z_j.m$.*

An example meta-graph can be seen in Figure 5.5. Importantly, this image is not of a concept lattice since it was not constructed from a single context and lacks a unique infimum. Rather, this image reports the hierarchical relationship of all concepts seen by the agent, along with behavioral information as represented by the nodes. Each node represents a concept (substate) feature $z \in \mathcal{Z}$, and its learned weight θ_i for z_i, given the agent's weight vector θ. The size of the node conveys the relative absolute magnitude of θ. Specifically, all non-negative weights were normalized between 0.0 and the maximum weight, and all negative weights were normalized between 0.0 and the minimum weight (so, the more negative a weight is, the larger it is). A node's color is blue if θ_i is positive or orange if negative. Thus, a large blue node signifies a strong positive weight, while a large orange node represents a strong negative weight, and smaller nodes are concepts with weights closer to zero. A directed edge between nodes shows the partial ordering of superconcept to (an immediate) subconcept, based on their intents. The length of an edge represents an approximation of the Hamming distance between the two concepts' intents.

In the meta-graph renderer, I apply physics-based repulsion and gravity such that dissimilar concepts exert greater force upon each other, and concepts with larger intents are pulled downward. Thus, nodes towards the top of the diagram tend to be more abstract (but, due to exerted forces, are not guaranteed to be), while those towards the bottom tend to represent more specific concepts. The highest node in the meta-graph corresponds

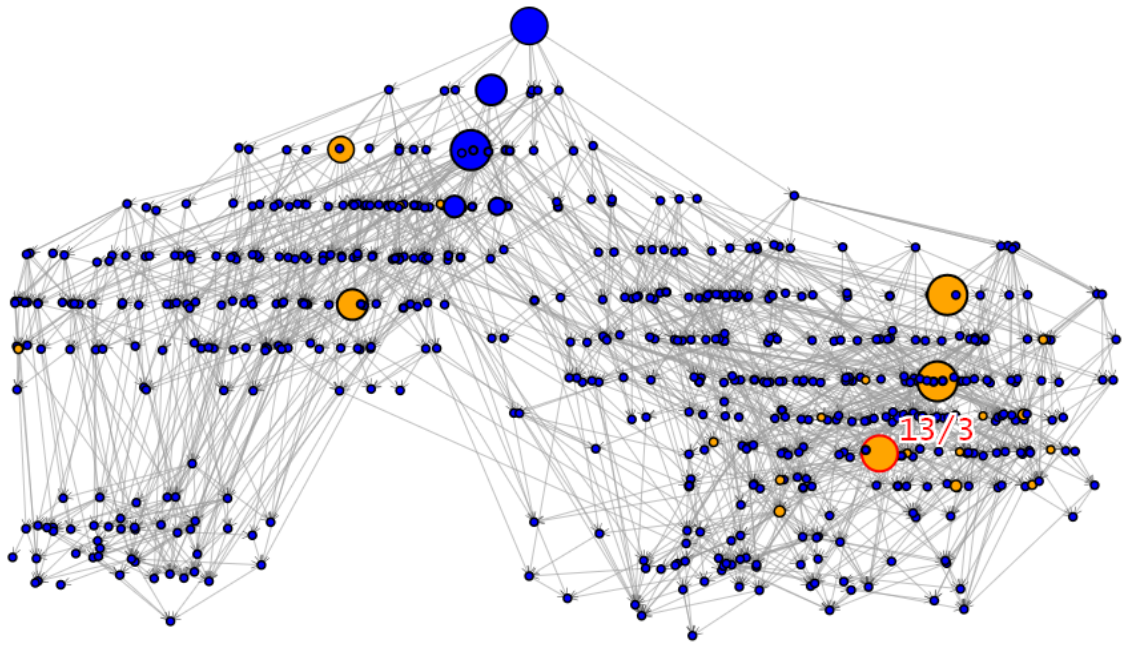


Figure 5.5: The meta-graph of concepts extracted in a single trial of 500 episodes for the Mushroom CB problem. The labels at the top describe the highlighted node (in red, at the lower right), which has 13 attribute-values and an object support of 3. The size and color of the node indicates CAHL has learned a large negative weight for this concept, as reported at the top: when this concept is observed for the "eat" action, it would add a weight of around -8.6212.

to the top concept, and it possesses the smallest intent (containing only those attribute-values common among all objects). I also enforce a minimum node diameter to keep them visible.

5.2.5.3 Explanations and Interpretability

The dual goals of explainability and interpretability have gained prominence and attention in recent literature [53]. While these terms are colloquially synonymous, I differentiate them, defining the former as more holistic and the latter more fine-grained. Specifically,

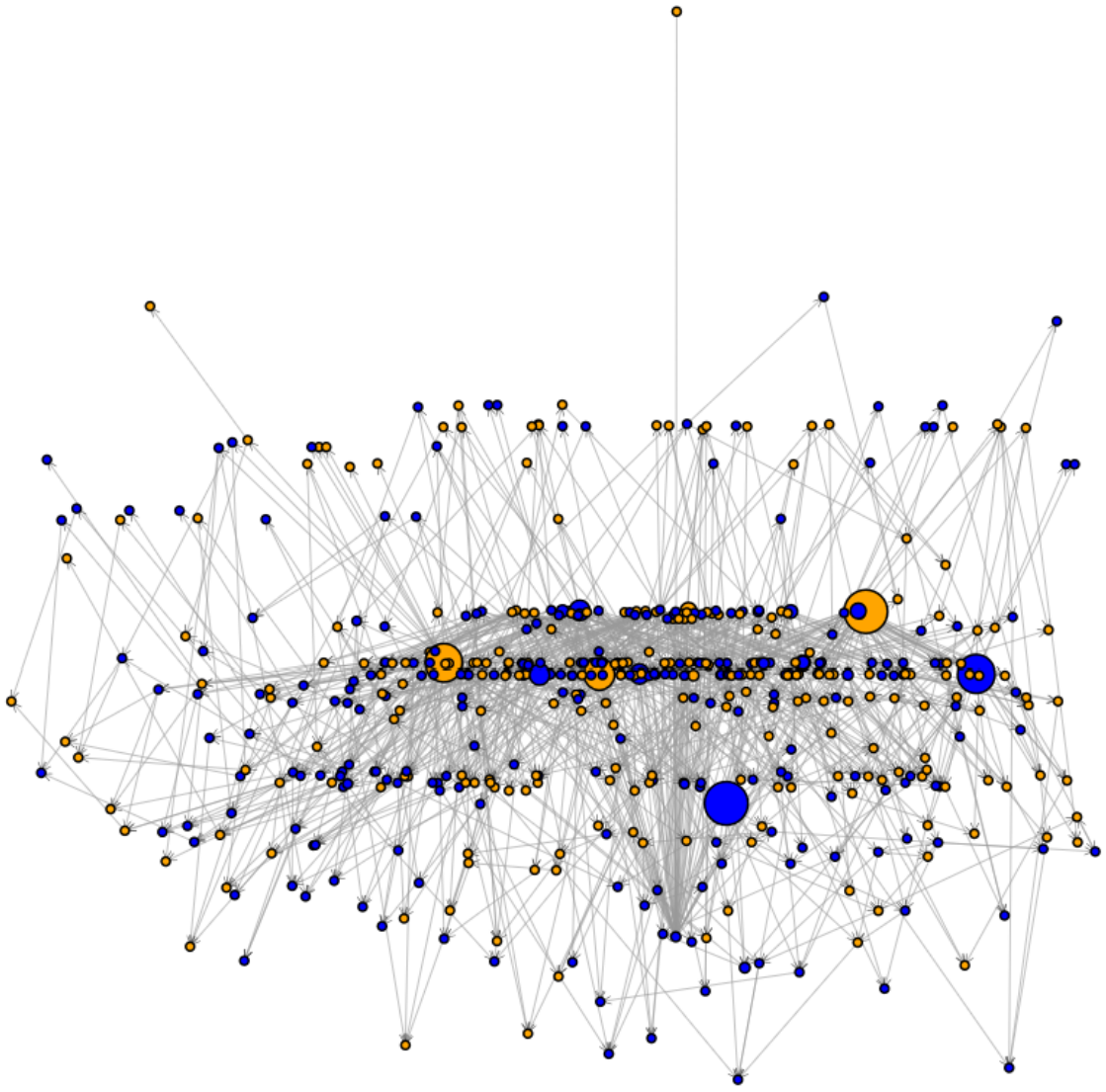


Figure 5.6: The meta-graph of concepts extracted in a single trial of 2000 episodes for the NetHack Monster CB problem.

with explainability I refer to an algorithm's ability to supply answers to "why" questions an observer might ask. Interpretability, alternatively, refers to a quality an algorithm possesses if its constituent parts can be easily characterized or understood upon human inspection. The subtlety of these terms comes into play for CBC in its anomaly reasoning capacity, explaining how it adapts to new objects (answering the question, "why did you select that action?"), and offering an interpretable knowledge base (learned weights for concept features).

The concept meta-graph of Figure 5.5 shows one trial of CAHL applied to the Mushroom domain, with a node highlighted (outlined in red) for a concept substate with 13 attribute-values and an object count of 3. At the top, there is a list of that concept's attribute-values, its object support, and the learned weight for each action. For the Mushroom domain, there is only one (parameterized) action, and thus one weight. In this case, the weight is negative, as reflected by the node's color, and is one of the larger weights learned, as evidenced by the node's size. This property embodies the interpretability of concepts as features: one can understand how an agent associates an action with a concept (positively or negatively), within the context of all other concepts. A key takeaway from this diagram is that most concepts have small (near-zero) weights, and are not relevant to the totality of its knowledge. Moreover, most concepts are weighted positively, with the effect of amplifying the impact of negatively weighted concepts when they are present

A general explanation of the results can, as a whole, be seen in terms of the handful of concepts with weights that dominate the direction of an agent's decisions. Recall that CAHL has a model for each arm—in this case one model for the "eat" action—using just the grounded features (of mushrooms), and that the separate joint model factors the

concepts derived from all mushrooms in the context. Thus, the visualized meta-graph captures how safe or dangerous the given context is overall (not the safety of individual mushrooms).

Clearly, concepts would offer only marginal benefit when a mushroom is easily categorized as edible or poisonous based on its raw features. However, my results demonstrate that concepts give a boost over L, indicating that they help guard against the selection of dangerous mushrooms. Specifically, concepts mitigate the effects of anomalies.

Consider an agent has been learning and encounters a new mushroom that appears safe to eat, based on its raw features, but is actually poisonous. Where L must make an assessment of an anomalous mushroom solely on its own merits, CAHL also takes into account the anomaly's conceptualization. Further suppose L incorrectly considers it safe, while CAHL recognizes the danger. It may be reasoned that, in order to have made this decision, CAHL must have learned negative weights for concepts associated with the anomaly, large enough to offset the positive weighting of the arm model. Thus, in generating an explanation of how CAHL appropriately adapts to unknown objects, one recognizes that the most negatively weighted concepts in the meta-graph must correspond to edge cases that appear safe on the surface but are actually bad. For the most positively weighted concepts, a similar conclusion can be reached vice versa, such that they help handle cases of anomalies that appear negative but are in fact good (and may be represented so, conceptually).

Finally, due to the poor performance of HL in the Mushroom domain (Figure 5.3), it may be concluded that concept features are more valuable than simply taking all arm features as the shared context. I contend that the abstract nature of concepts plays a de-

cisive role in their ability to more quickly adapt to uncertain objects. As latent groupings of features, concepts help characterize novel objects more readily, such that an anomaly would have to be wholly unfamiliar to not receive some initial conceptual bias. The interpretation of anomalies done by CAHL, then, helps explain the agent's decisions not just in terms of any one object, but of that object contextually at several levels of abstraction.

5.3 Concept-Aware Reinforcement Learning

I now turn to decision making in which an agent's actions have a direct effect on the environment, reinforcement learning (RL). The CBC methodology demonstrated the viability of concepts as a vehicle for increasing generalization and transfer across new and anomalous objects; however, it did so under the assumption that successive contexts lacked any causal connection. As opposed to CB problems, in RL the agent must consider feedback from the environment, where the next visited state depends on the action just taken, determining long-term reward based on this sequence of transitions. An agent operating from the vantage of a Markov decision process observes its state, takes an action, and transitions to a successor state as a direct consequence. Because reward in RL depends upon the transition and successor state, the expected cumulative reward is recursively dependent on all future actions. Thus, in this section I seek to uncover the applicability and properties of CAFE when used to abstract states over longer periods of time, when each successive decision influences the next.

5.3.1 Concept Features & Temporal Difference

Concept-aware reinforcement learning (CARL) encompasses algorithms that rely on CAFE to approximate the action-value function ($\hat{Q}_\theta$). In this dissertation I consider temporal difference approaches, in which a value function is learned by bootstrapping from experience (as described in Section 2.1). I implement and analyze two novel approaches: concept-aware Q-learning (CAQL) and concept-aware SARSA with eligibility traces (CASARSA(λ)).

In reference to VFA, I refer the reader to Section 2.1.2, in particular Equation 2.6. Using CAFE as the VFA technique is more straightforward for CARL than CBC, since there is no separation of objects versus side information, and I use one model (for the action-value, rather than one per action). As before, I consider the linear case, where an agent's goal is to learn a parameter or weight vector θ. On each time-step, the state s and action a serve as the input to CAFE by which I attain the $\phi(s, a)$ featurization. Note that this process is identical to CAFE as defined in Definition 5.1, just with features factored across each action, as is standard for action-value function approximation [149].

I include pseudocode for CAQL in Algorithm 1. I apply CAFE to an action-value function approximation scheme, updating as in gradient descent Q-learning [113]. For CASARSA, I follow the same principle, using CAFE in the function approximation for gradient descent SARSA(λ) [132].

In terms of anomaly reasoning, CARL methods can be viewed as implementing HIGHLIGHT, ADAPT, and UPDATE. The HIGHLIGHT process is handled by computing the function approximation based on the concept states present, and ADAPT selects an ac-

Algorithm 1 Concept-Aware Q-Learning

function CAQL (OO-MDP $\langle \mathcal{S}, \mathcal{A}, T, R, \gamma \rangle$, terminal condition $\tau : \mathcal{S} \to \{0, 1\}$)
  Set t_{max}, maximum time-steps for the episode
  Get the initial state s_0, initialize $\hat{Q}_\theta$, weight vector θ, and learning rate α
  $t \leftarrow 0$, $\mathcal{Z} \leftarrow \varnothing$
  while $t < t_{max}$ and not $\tau(s_t)$ **do** ▷ while episode has not terminated
    $(\mathcal{O}, \mathcal{M}, I) \leftarrow \rho(s_t)$ ▷ convert state to formal context
    $\mathfrak{B} \leftarrow \Gamma(\mathcal{O}, \mathcal{M}, I)$ ▷ build concept lattice
    $\mathcal{C} \leftarrow \textsc{Count}(\mathfrak{B})$ ▷ convert formal concepts to abstract concepts
    $\mathcal{Z}_{s_t} \leftarrow \Psi(s_t, \mathcal{C})$ ▷ generate the concept substates
    $\mathcal{Z} \leftarrow \mathcal{Z} \cup \mathcal{Z}_{s_t}$ ▷ Subsume new concepts, expand ϕ for each new element
    $\hat{Q}_\theta(s_t, a) \leftarrow \theta_t^\intercal \phi(s_t, a),\ \forall\, a \in \mathcal{A}$ ▷ compute the Q-value of each action
    Select a_t, e.g., based on $\hat{Q}_\theta(s_t, a_t)$ using a Boltzmann or ϵ-greedy policy
    Execute a_t, observe successor state s_{t+1} and reward r_t
    $\theta_{t+1} = \theta_t + \alpha_t \big(r_t + \gamma \max_{a \in \mathcal{A}} \hat{Q}_\theta(s_{t+1}, a) - \hat{Q}_\theta(s_t, a_t)\big) \big(\nabla \hat{Q}_\theta(s_t, a_t)\big)$
    $t \leftarrow t + 1$

tion according to a policy based on the action-value it is approximating. Both procedures use the appropriate parameter update rule as defined in Equation 2.7.

Complexity

Considering computational complexity, CAQL is similar to QL, which is $O(|A|)$ per-time-step in general, or $O(\log |\mathcal{A}|)$ if using a max-heap priority queue [146], with an additional term for concept generation. The parameter-wise updates are linear in the number of features, so approximate QL has per time step complexity of $O(d|A|)$ for d features.

For CAQL, generating formal concepts must be done per state visited. Recall the CAFE uses IN-CLOSE2 to extract formal concepts. Its asymptotic worst-case time complexity has been shown to be $O(|\mathfrak{B}||\mathcal{O}||\mathcal{M}|^2)$ [164]. To obtain the concept lattice, I use Lindig's UPPERNEIGHBORS algorithm that proceeds bottom-up, starting with the zero concept found by IN-CLOSE2, recursively finding the parent concepts of each given

concept, with worst case complexity of $O(|\mathfrak{B}||\mathcal{O}|^2|\mathcal{M}|)$ [93]. Therefore, one obtains a single-step concept formation complexity of $O(|\mathfrak{B}||\mathcal{O}|^2|\mathcal{M}|^2)$. Since every state maps to exactly one set of concepts, each state needs to have its concepts computed only once (the first time it is visited) and cached. The mapping of states to concepts (and the resulting concept states) can also be stored in a constant-time lookup table. I note that $k|\mathfrak{B}_t| = |\mathcal{Z}_t|$, with $k > 1$ only when an expert has specified held-out attribute-values (meaning there will be multiple groundings of a formal concept to a state), which merely increases the number of concept states by a constant factor. Thus, there is an upper limit to the number of times concept generation must be performed for a given domain: $|\mathcal{S}|$.

Given a maximum of t_{max} time steps and the set of all concept substates $\mathcal{Z}$, CAQL has a total computational complexity of $O(t_{max}d\log(|\mathcal{A}|)+|\mathcal{Z}||\mathcal{O}|^2|\mathcal{M}|^2)$. Due to the assumption of task relatedness in the domains I examine, I observe that $|\mathcal{O}|^2, |\mathcal{M}|^2 < |\mathcal{Z}| = d \ll |\mathcal{S}|$. Thus, the added increase in complexity is more reasonable than the worst case makes it seem, and in principle grows linearly with $|\mathcal{S}|$. Similarly, space complexity increases only by $|\mathcal{Z}|$, assuming lattices are stored for caching purposes. It is also important to note that the purpose of making temporal difference learning algorithms concept-aware is not to improve upon their efficiency, but to jumpstart performance in future tasks by paying some computational work upfront to capture abstract representations.

5.3.2 Transfer of Concepts

The nature of VFA in RL facilitates generalization. VFA forces an agent to learn behaviors represented by a surface (the value or action-value function) in a vector space, relying

on features such as state factors as opposed to states themselves as discrete symbols. An agent reuses knowledge such that previously seen features appearing in new states for novel tasks or domains already have a learned weight associated with them [149, 154]. Transfer for RL in this case consists of preserving the weight parameters θ from solving one source MDP to the next (target) MDP.

For VFA paired with CAFE, one observes that transfer is granted the added benefits of abstraction, extensibility, and interpretability as discussed in Sections 5.1.3 and 5.2.5. In addition to transferring learned weights, the canonical orderings of attribute-values must also be transferred (to preserve the ordering of feature space, $\mathcal{Z}$). The cached state-lattice mapping can also be retained to speed learning across episodes.

Transfer inherently biases learning by associating behaviors with aspects of the state, so the similarity of source to target tasks can greatly affect the success of transfer. Tabular representations are more challenging to transfer; in general, they must retain the table of Q-values which are only applicable if the same index, a state-action pair, is visited in the target MDP [154]. For the action-value function approximation of CARL algorithms, successful transfer depends on the ability of concepts to capture meaningful actions in the source task that also apply in the target task.

Algorithms using tile-coding, which I henceforth refer to as "CMAC," also use linear VFA. However, CMAC approaches utilize pre-designed static tilings over state space (thereby requiring the full range of factors in state space to be known *a priori*). To transfer the tilings from source task to target task when the underlying state space is larger (for example, additional state factors), I apply a straightforward "behavior transfer" activation mapping procedure used previously in literature [153]. This strategy has us copy the

learned weights to the new tilings whose old tiling was a subset of the new one (possible again since it is assumed one knows the full range of factors). To handle new features, weights for tiles activated by existing factors are initialized to their weight copied in, else zero. Note that the action space remains constant in both experiments (if it increased from source to target task, transferring tiles would required additional engineering to design a solution accounting for the new actions). This difficulty in transferring learned weights in linear VFA is a key motivation for CARL, as a mechanism that inherently scales to new features and transfers in the weights for all those known.

In the case that training biases certain behaviors that are not useful to the target domain, an agent may suffer *negative transfer* and have to unlearn tendencies before coming to a solution, whereas an agent starting from scratch may learn to solve the task faster (yet of course, this newly initialized agent is not general). Thus, an overall challenge of transfer learning is thus to ensure the agent is general enough to adapt such that negative transfer does not outweigh the benefit of transferring knowledge. There is not a consensus when it comes to designing source and target domains, primarily because the degree of task relatedness between source and target is difficult to measure [154, 155]. In my experiments, I address this issue by train the agent on a task universe, with sample MDPs pulled from a distribution related tasks, but target a universe with an overall harder, more complex goal. The following sections elaborate on this idea and the specific train/test transfer methodology employed for each examined domain.

5.3.3 CARL Methodology

As described in Section 2.1.1, an OO-MDP is a natural and extensible way to express complex RL and planning domains. As the number of objects increases, the size of the state-action space increases combinatorially, approaching intractability for methods without VFA. Thus, I evaluate CARL algorithms on such domains, taking advantage of their states' definition as sets of objects to apply CAFE straightforwardly. In each domain investigated, the source and target tasks are identical except for variations at the object level (different attribute-values like color and shape, increased numbers of objects), emulating the kind of variety a robot would be expected to encounter in a natural environment.

I seek to examine the effect on transfer of using (abstract) concept features in comparison with grounded features. Thus, I elect to contrast CAFE with CMAC, a standard technique for linear VFA. CMAC works by specifying regular tilings across all state factors in state space, using each cell as a feature. As a sparse method based on the coincidence of state factors, I consider CMAC featurization to be directly comparable to CAFE, except that CMAC focuses solely on grounded features, whereas CAFE considers abstract clusters. I contrast CARL methods with two CMAC ones: CMACQL and CMACSARSA(λ). In my experiments, their performances are assessed in reference to baseline methods learning from scratch (QL and SARSA(λ)).

5.3.3.1 Domains & Tasks

I consider two domains, Traffic Light and Cleanup, in which I measure the benefit of learning and transferring concepts from a source task to a target task.

In terms CAFE settings, I introduce new "has" indicator attribute-values. Taking each object in a state, CAFE appends one new attribute-value per attribute of the object's OO-MDP class. For example, an object with just a color and a shape would have the two literal color and shape attribute-values, as well as two new attribute-values indicating that the object has a color and has a shape (in general). This process allows, for example, formal concepts to be generated beyond specific values, capturing the more general notion of objects that have any value of that attribute at all. The concept lattice in Figure 5.7b includes instances of these attribute-values. In all experiments, I leave the minimum object support at zero.

Traffic Light

The Traffic Light (TL) domain presents a simplified driving scenario in which the agent assumes the role of a vehicle waiting at an intersection. The agent must decide at each time-step whether to go or to wait. Each state consists of a traffic light (either green or red), some number of cars of various colors, and an agent that tracks its progress. The traffic light always begins as red, and whenever the agent waits, the traffic light switches color. If the agent chooses to go when the traffic light is red, the episode terminates. If the agent elects to go when the traffic light is green, the agent's progress is incremented; the episode terminates and the agent receives a positive reward once the progress counter reaches some predetermined threshold number referred to as the *horizon*. Thus, the goal of TL is to go when the light on a traffic light is green, such that the optimal policy is always to wait once for the traffic light to turn green and then go n times until the horizon

is reached.

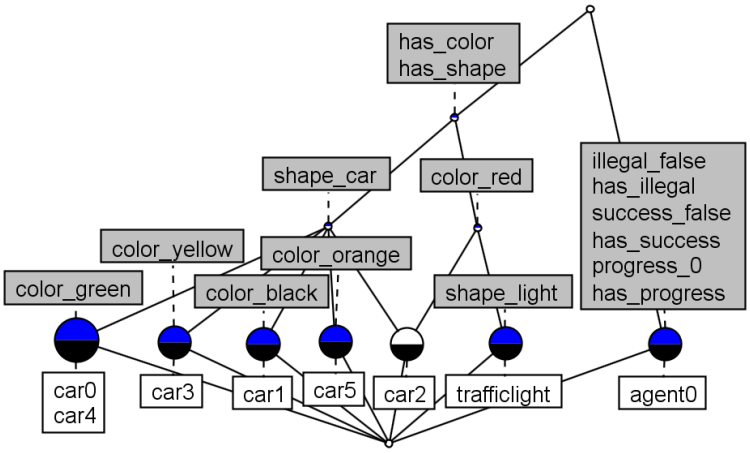
I use TL-n to refer to a specific instance of a TL task with a horizon of n. Each task may have any number of cars, each with the shape "car" and one of among ten colors (including red and green) selected uniformly at random. Transitions are deterministic. Reward is sparse, with $+1$ given upon reaching the progress threshold.

My goal in experimenting with TL is to understand the value of CARL over standard approaches for transfer in a minimal MDP example. The inclusion of cars adds a form of noise to the domain, straightforwardly increasing the degree of task complexity by including objects that are ultimately irrelevant to the task's solution. Introducing this confounding element makes learning harder for tabular methods, but also challenges CARL algorithms by adding a substantial degree of knowledge that is wasteful to transfer. Ultimately, I seek to show that CARL algorithms surmount these obstacles to learn the importance of the most relevant concepts (that "green traffic light" has higher value than "green car" or just "green").

Cleanup

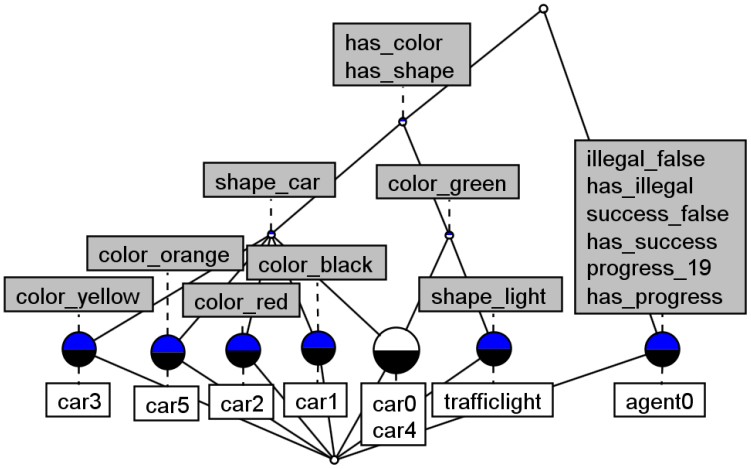
The Cleanup domain presents the agent in a gridworld environment composed of rooms, blocks, and doors. The aim of Cleanup is to simulate a robot with the goal of tidying a house by putting blocks where they belong, similar to the game of Sokoban [95]. Room objects may have any color and take any width or height. Blocks possess a color, shape, and x-y coordinates; they must be pushed and pulled into certain rooms. Door objects have the same attributes as blocks, and they are always placed along the wall between

(a) Example starting stat with a red light, no progress, and six cars (two green, one each of orange, black, red, yellow).



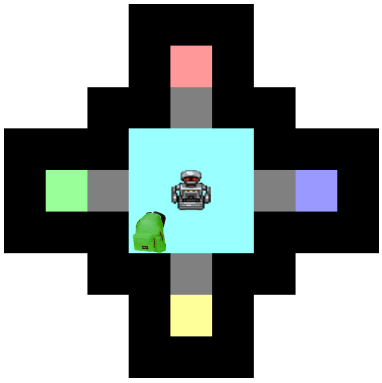
(b) The concept lattice derived from the state to the left.

(c) Example penultimate state from the same task of Figure 5.7a, now with a green light, and nearly finished progress.

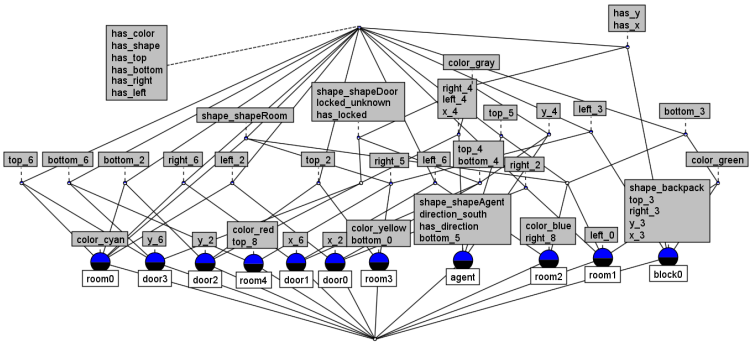


(d) The concept lattice derived from the state to the left.

Figure 5.7: Example states from the Traffic Light domain and their concept lattices.



(a) Example starting state.



(b) The concept lattice derived from the state to the left.

Figure 5.8: Example state from the Cleanup domain's Cardinal Closets task and its concept lattice.

two rooms. The agent also has shape, color, x-y coordinates, and an attribute for the specific direction it faces. The agent may move one step in the cardinal directions, or a take a "pull" action, allowing the agent to swap positions (and its direction) with a block if one is immediately in front of the agent. Upon moving in the direction of a block, the agent pushes the block forward if possible (not further blocked by a wall or another block).

I describe the particular task universe of Cleanup in the experiments as Cardinal Closets (CC-n). In CC-n, the agent exists in a central room containing n blocks each of a random color in a random position, and four closets of different colors that are connected to the central room in the cardinal directions. The goal is to maneuver one specific block into the room of the matching color. Variant domains can be described by including more blocks (of different colors). Transitions are deterministic. If the agent's action results in no state change (i.e., a self-transition of $s_t = s_{t+1}$) then it receives a reward of -0.0001, otherwise it gets a reward of only 0.0 until a terminal goal state is reached ($+1.0$). CC in particular is challenging because the optimal policy for agent navigation changes completely based on the color of the goal block (i.e., if the block is red then the agent must manipulate the block towards the north, or south if the block is yellow). So, in some sense, CC wraps four tasks in one. Moreover, the nature of the central room means that an agent continually runs into corner cases and states in which a block is against a wall, requiring the agent to maneuver precisely to pull the block away from the wall into a position where it may be pushed. It is important to note that the size of these domains is considerably different, as each added block grows the state space combinatorially (e.g., CC-1 has on the order of thousands of states; CC-2 has tens of thousands).

5.3.4 CARL Results

I follow a training-testing pattern for evaluating transfer. An agent first learns on a set of training domains to solve a source task and then is evaluated on another set of domains to solve a target task. Domains for both the source and target tasks are drawn randomly from the same distribution, though the target is an alternative, more complex variant of the source task.

The algorithms I assess are QL and SARSA(λ), as the tabular non-transfer baselines, as well as CMACQL, CMACSARSA(λ), CAQL, and CASARSA(λ). The CMAC pair transfer experience in the form of weights associated with tiling of state space, while the latter two transfer concept feature weights (θ and the canonical orderings of features serve together as the knowledge base K for each agent). During training, I record the agent's knowledge K upon completing a trial, and then freeze and transfer the averaged value of K to the target task, reloading the transferred K for each of those evaluation trials. For the former two algorithms, transfer is straightforward in that the entire Q-table is persisted from task to task. For the CMAC algorithms, I use eight tilings and pre-configure them with the number of objects and the range of possible values for every attribute. In assessing my results, I also confirmed that each of the CMAC and CARL methods with transfer improved over an identical version training from scratch (without transfer) when run on the same experiment trial (same random seed).

In the plots of the results, I report the 95% confidence interval as represented by the shaded regions around each graphed line. These experiments also used the same parameters: a $\gamma = 0.99$ discount factor following an epsilon-greedy exploration policy

with $\epsilon = 0.10$ and constant $\alpha = 0.01$ learning rate. SARSA and CASARSA employed a $\lambda = 0.75$ trace decay.

5.3.4.1 TL Results

Training consisted of 100 trials on a TL-5 task containing six cars. Evaluation was done over 50 trials on the TL-10 task, again with every state containing six random cars. All algorithms were given 300 episodes, each with a maximum of 100 steps.

The results of training in TL are shown in Figure 5.9. In regarding the cumulative reward, it is clear that every algorithm eventually solves the problem. The number of steps reveals finer-grained functional differences among them. In general, the SARSA methods are more stable, in large part because their on-policy nature combined with eligibility traces makes them more effective at control, avoiding transitions that lead to early termination. QL methods explore more, such as in CAQL where more episodes terminate early (exploring the effect of running a red light), and later episodes take longer than necessary (exploring waiting at various points up to the horizon). Both CMAC algorithms readily solve the task; they rely on the simplicity of the domain, where the optimal action depends linearly on one state factor (the light's color). By contrast, the CARL algorithms take longer: they have more features to learn in general, but must also disentangle the concept of green lights from green cars (and the color green in general). This behavior is as anticipated: concept features take time to learn, but increase performance when deployed to handle novelty and anomalous features.

Figure 5.10 presents the evaluation performance after transfer on the TL-10 task.

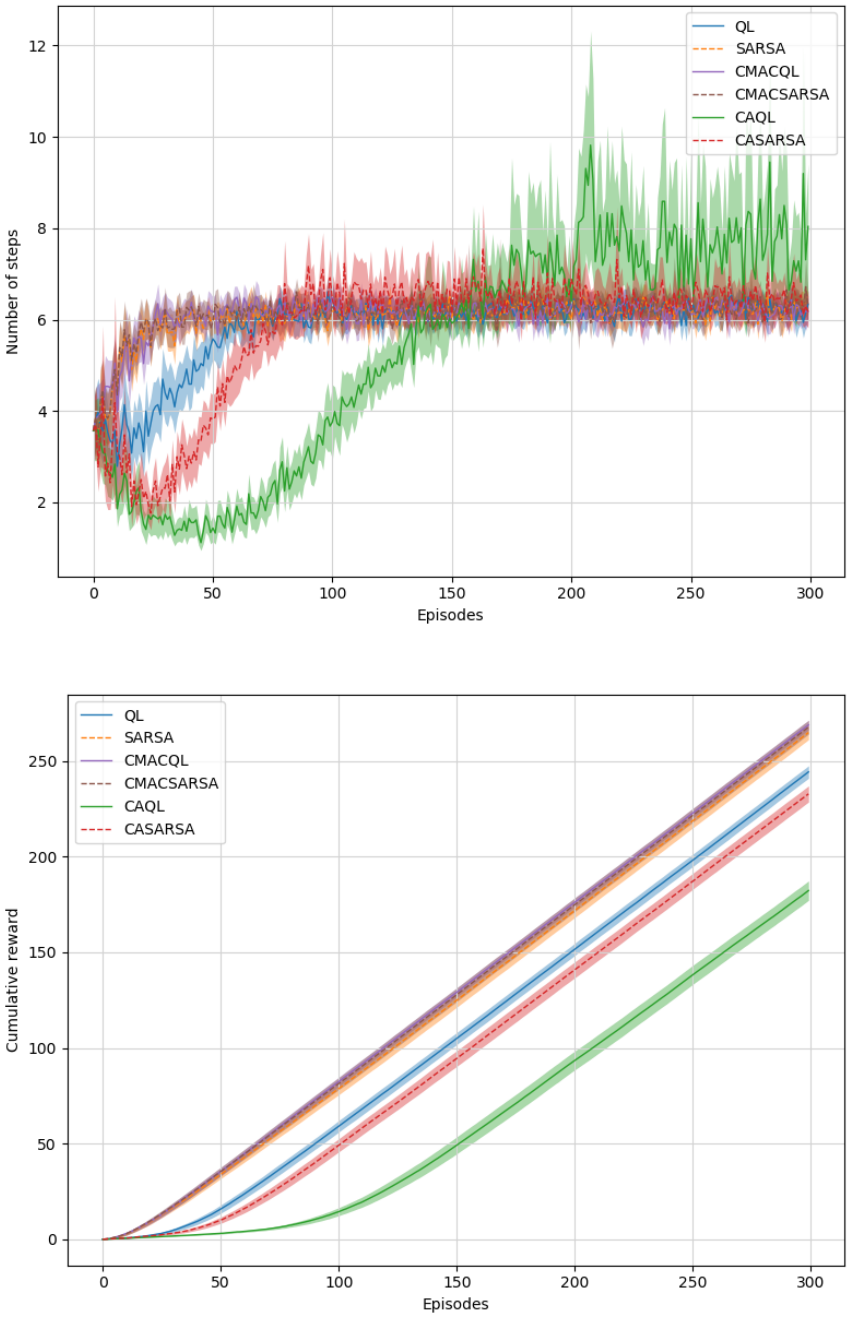
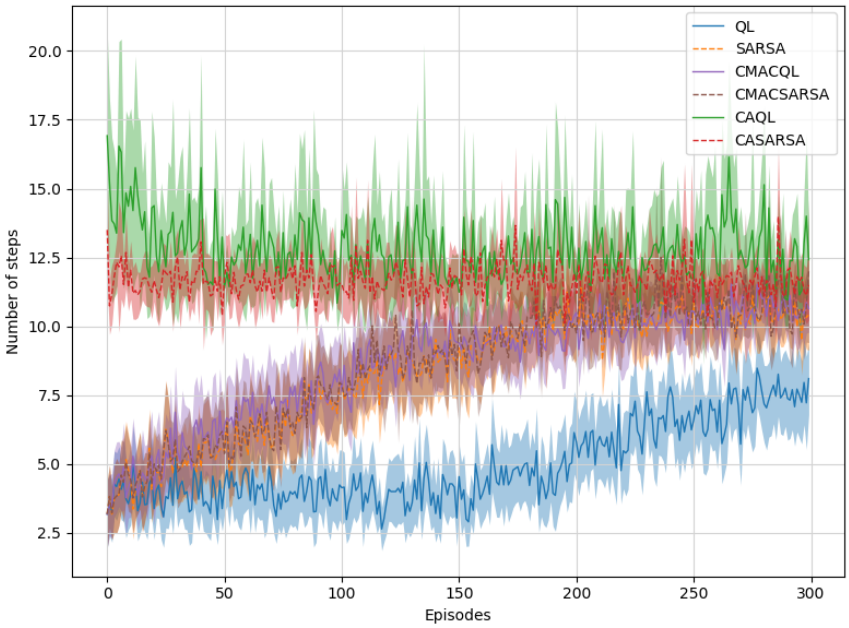


Figure 5.9: Training results for 100 trials of TL-5 with six cars.



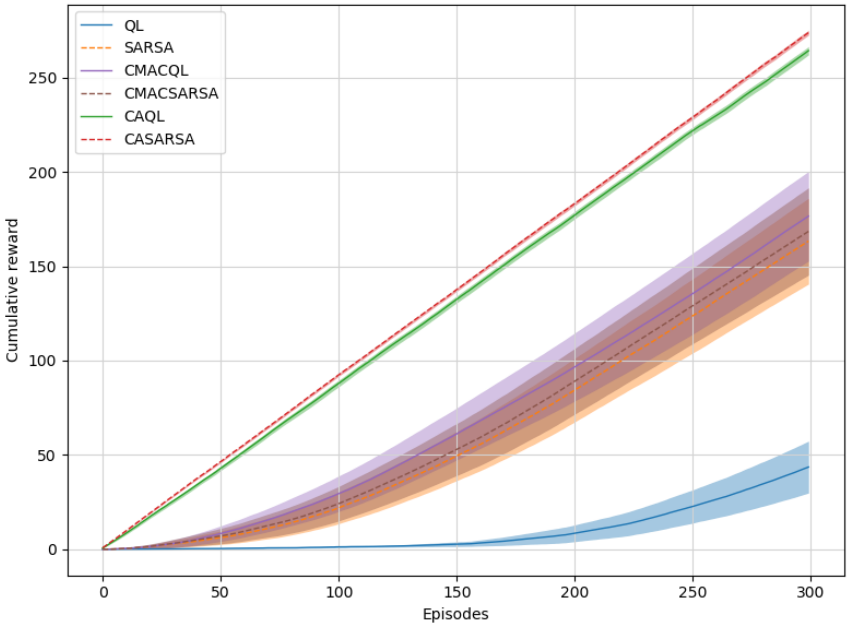


Figure 5.10: Evaluation results in terms of the number of steps and the cumulative reward for 50 trials of TL-10 with six cars, transferring knowledge from the TL-5 task.

Here, it may be observed that the concept-aware methods receive an immediate jumpstart transfer, solving the domain from the first episode onwards. This behavior is seen clearly in the plot of the number of steps, where CAQL and CASARSA take around ten to fifteen steps (and reach the goal), where the other methods terminate prematurely.

It can be seen that the CMAC tilings, representative grounded features, fail to transfer as well as the more abstract concepts. In fact, they perform only as well as the non-transfer baseline of SARSA. I offer an explanation: the grounded nature of the CMAC features hinders their ability to transfer. Specifically, they learned knowledge of how to act by correctly associating large weights for "go" when the traffic light was green; however, such tilings also included the progress attribute-value of the agent, so beyond five steps the algorithms did not know how to react. Thus, the CMAC algorithms terminate early and as often as SARSA, which is learning from scratch.

The CARL algorithms leverage their knowledge of abstract concepts to receive this boost; part of my motivation for investigating TL is to demonstrate this effect clearly, in a minimal example. Inspecting the weights associated with concepts after training reveals how they achieve transfer. In particular, the feature-action pair with the largest weight is for the concept state corresponding to $\{$shape:light, color:green$\}$ and action "go" with a value of 7.625. Symmetrically, the pair with the smallest weight is $\{$shape:light, color:red$\}$ and "go" with a value of -7.388. The concept-aware algorithms find approximately 12 concepts per state (hence, 24 feature-action pairs), with a total of 61 concepts (the final $|\theta| = 122$) across all states. Asymptotic performance in TL is also increased when using concept-aware algorithms, particularly for SARSA which follows a more consistent policy than the QL approach.

Overall, the TL experiments highlight, in a constrained example, the ability of CAFE to capture high-level feature clusters for VFA, facilitating knowledge transfer to a more challenging task. As witnessed with CBC, I illustrate how concept features permit anomalies (in this case, new states closer to the horizon) to be interpreted correctly. Thus, the results indicate that concept-forming temporal difference agents can successfully adapt based in terms of choosing what action to take in an RL setting.

5.3.4.2 CC Results

I now consider CARL in a more complex MDP scenario, with transfer from one-block tasks (CC-1) to those with two blocks (CC-2). For setup, all algorithms are given 1000 episodes each trial with a maximum of 1000 steps. Training was done on 100 trials for CC-1, and 50 trials of CC-2.

The results of training are shown in Figure 5.11, while Figures 5.12 and 5.13 show the performance after transfer on the evaluation domains. In general, the tabular methods take longer to learn (as expected), and QL does not find a reasonable policy in the episodes allotted. CMACSARSA performs best, quickly finding a policy that minimizes the number of steps to the goal state. However, this result is at the expense of exploration (as will be seen in the evaluation results), with the result it will learn less information to transfer. CMACQL and the concept-aware algorithms perform roughly the same: they take around 600 episodes to learn a decent policy. In contrast with CMACSARSA, these algorithms require more exploration before homing in on a solution. They also suffer from larger variance in the number of steps in later episodes, again due to a larger propensity

for exploring.

In evaluation, one can see the ability of the concept-aware algorithms to persist and transfer the knowledge they acquire more effectively than their CMAC counterparts. In Figures 5.12 and 5.13 it is evident that concept-aware algorithms more quickly adapt to the greatly expanded state space of CC-2. Their performance much more closely matches that for CC-1, whereas QL and SARSA suffer from the increased size of the domain. Both exhibit a jumpstart boost in performance from transfer. In comparison to CMACQL, SARSA, and QL, the CAQL and CASARSA also yield better asymptotic trends. CMACSARSA benefits from transfer as well, approximating CAQL's performance, but in a more muted way than its more direct analogue of CASARSA. CMACQL also occasionally exhibits negative transfer, in that it can underperform relative to another CMACQL agent without any transferred weights (in contrast to CMACSARSA and the CARL methods, which always improve with transfer).

The highest valued feature-action weights are for concepts associated with a block being in the door adjacent to the block's respective goal room, paired with the navigational action that transitions to a goal state (e.g., "north" if a red block is inside the door to the red closet). Other highly valued weights include those for concept-action pairings that align the block with the door and goal room, or otherwise manipulate it out of a corner. In total, the concept-aware algorithms find 466 distinct concept states, and thus 2330 feature-action pairs (the final $|\theta| = 2330$) across all states from all domains. Each state in CC-2 has on average 40 concepts. Thus, the space of concept-actions is considerably smaller than the state-action space of CC-2, explaining why action-value function approximation with CAFE grants both the benefits of jumpstart transfer and increased asymptotic perfor-

mance. The ability of CAFE to achieve a condensed, semantic compression of state space when objects' attributes can take many possible non-numeric, qualitative values (such as color and shape), as indicated by these results, is promising for further development of explicit anomaly reasoning for planning and reinforcement learning.

5.3.5 CARL Discussion & Analysis

CAFE is not always an ideal approach, but TL and CC highlight cases in which it can excel. In the TL tasks, there always exists some feature corresponding to an action that matches the optimal policy: the concept corresponding to the traffic light. That is, there is always a concept present for the light being green, in which the desired action is to go, or the light being red, in which the best action is to wait. An interesting side effect is that related concepts such as "color:green" will also be given a slight bias in favor of "go," mirroring the associations of concepts that occurs in human learning (having seen red stop signs our entire lives, we are more wary about proceeding whenever given a red light). When a concept does exist that directly signals a good behavior, CAFE will certainly detect and exploit it, as CMAC will do when a grounded state factor does the same. Even when they are presented with anomalous objects, CAQL and CASARSA are robust to both the threshold and the presence of cars, thereby effectively and correctly ignoring them. It is the case that an observer can inspect what CARL methods learn to see this process reflected: for the concept substates less relevant to the goal (e.g., any related to cars), their learned weights near zero.

The CC tasks shed light on the difference between CAFE and CMAC in terms of

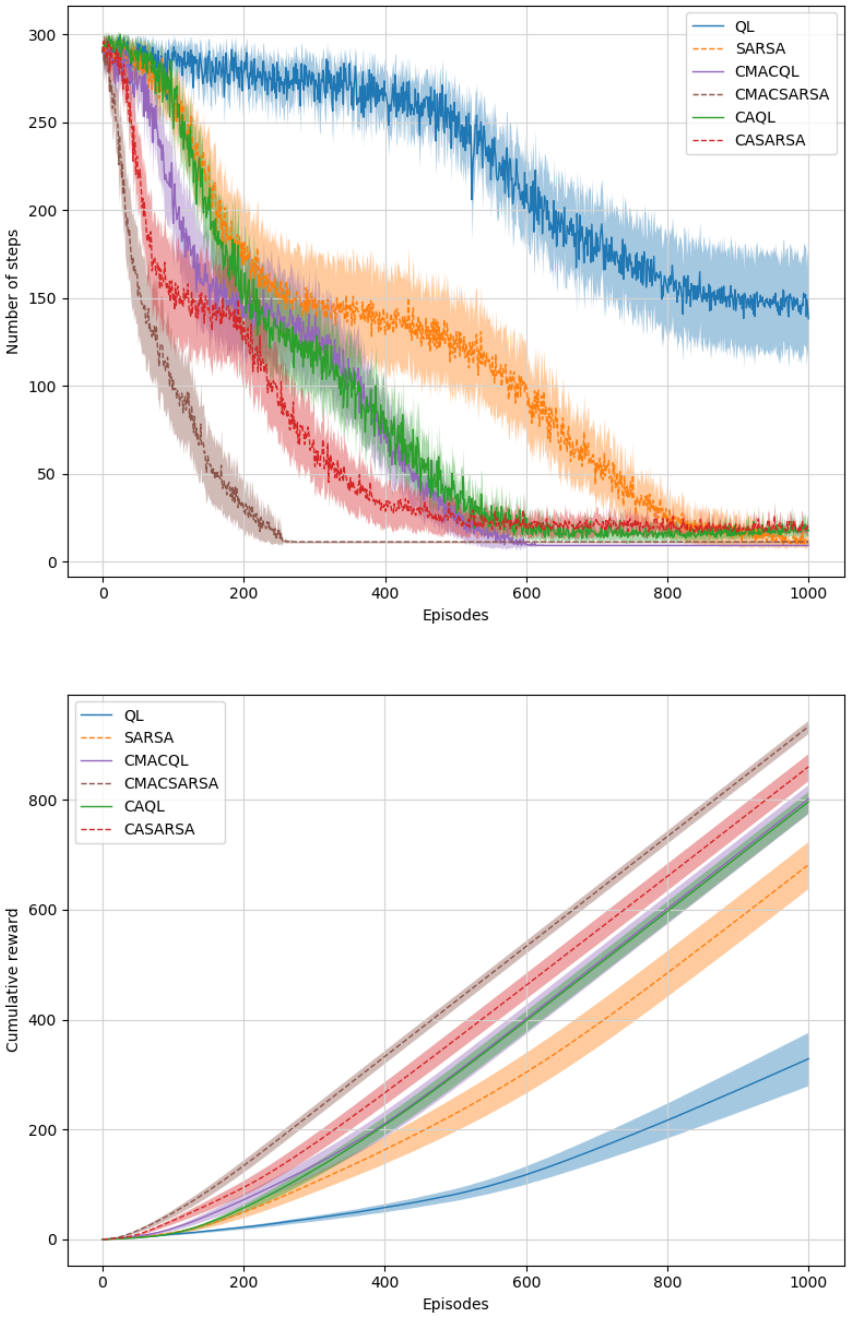


Figure 5.11: Training results on CC-1 for 100 trials.

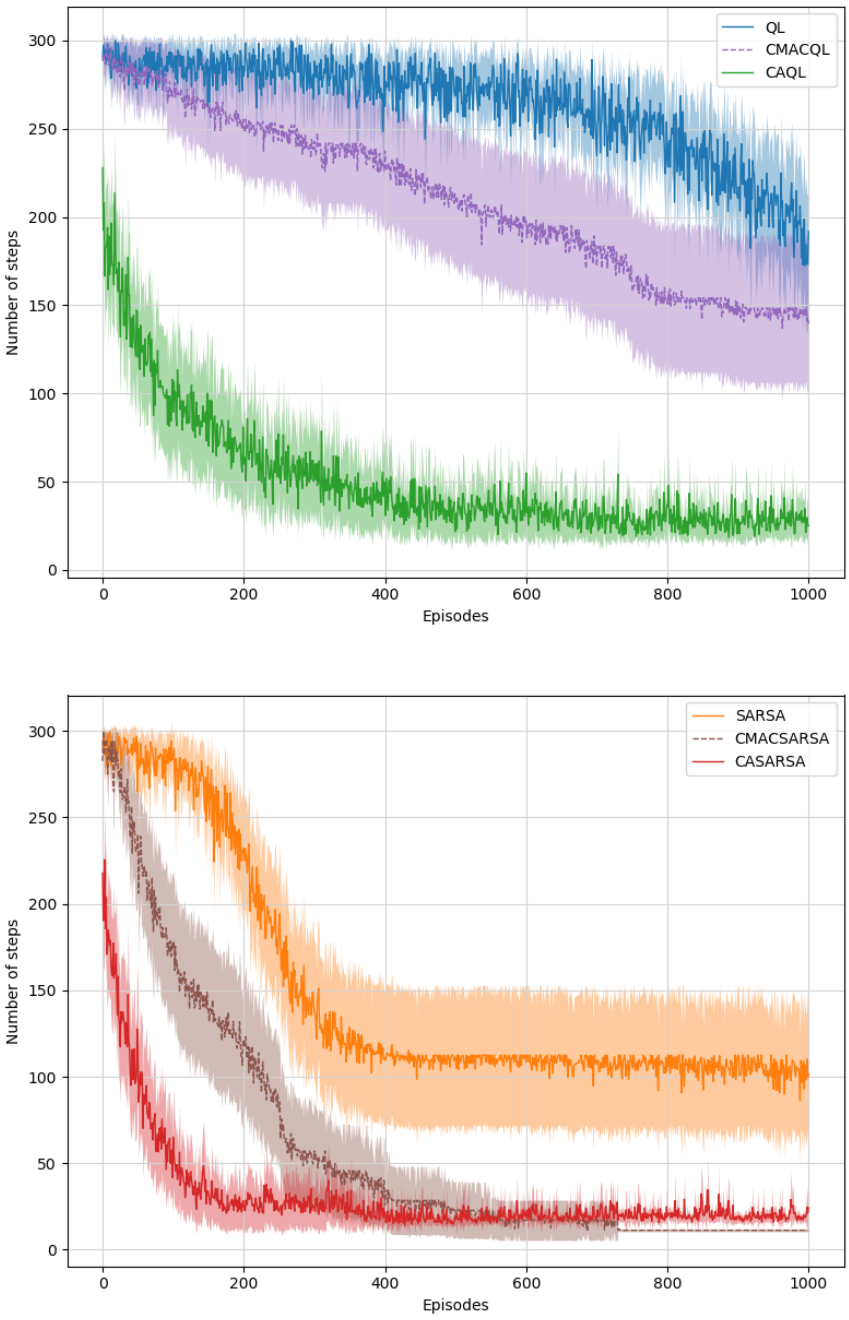


Figure 5.12: Evaluation results in terms of the number of steps for 50 trials of CC-2, with transfer from training on CC-1. I split the QL and SARSA into separate plots so the differentiation among them is more readily apparent.

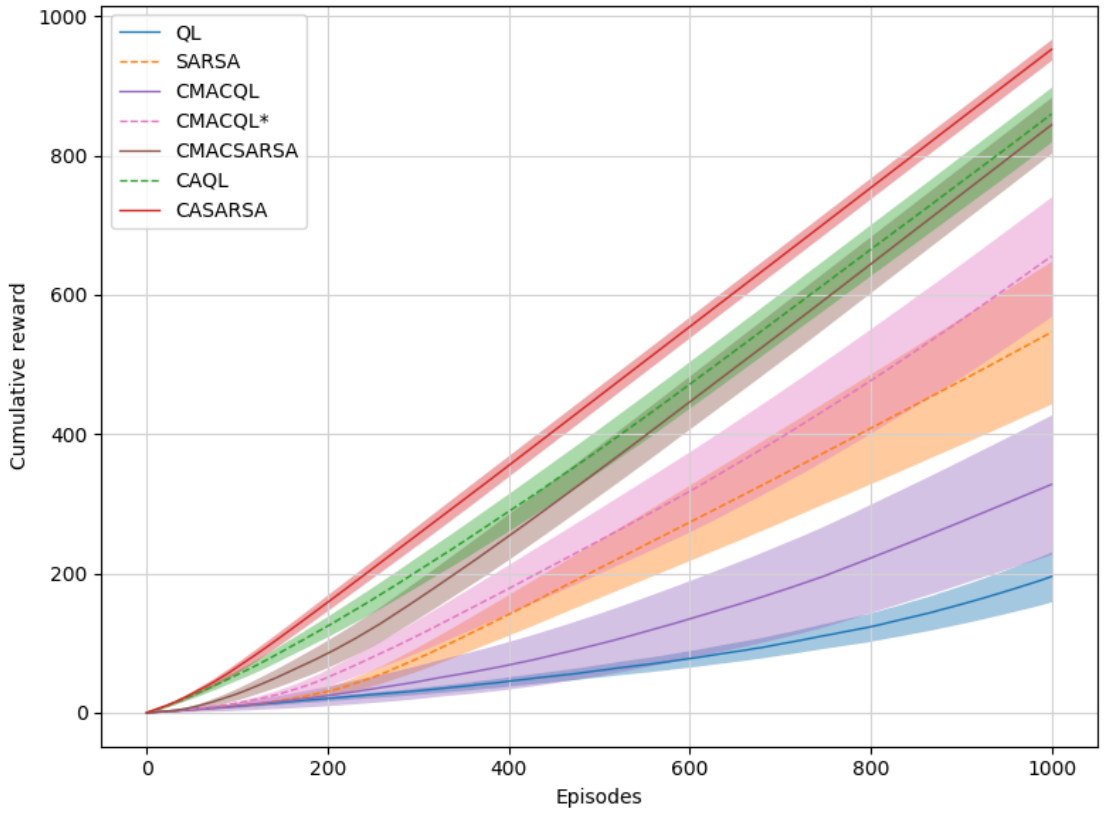


Figure 5.13: Evaluation results in terms of cumulative reward for 50 trials of CC-2, with transfer from training on CC-1. In comparison with non-transfer methods run on the same trials, I found that CARL methods always receive a boost. However, observe that CMACQL exhibits negative transfer. In the graph, I include CMACQL* as the version of it *without* transfer. CMACQL* learns from scratch, and because it significantly exceeds the performance of CMACQL trained on CC-1, one can clearly see the impact of negative transfer. This pattern was not observed with CMACSARSA, indicating that the transfer bias may more greatly affect an off-policy approach.

transfer. Significantly, in these tasks there is no single feature (as in TL) that guides good behavior. That is, a CC state possesses no one object or feature that acts like the traffic light's color, as an indicator of the optimal action. Agents must instead rely on the linear combinations of many weighted features to find the appropriate action-value functions. Thus, I view CC as more realistic of the challenges an agent might face in RL and learning transfer.

Upon inspection, concepts related to the color of the block are clearly the strongest contenders to be indicative of behavior. Indeed, looking at the learned weights reveals that color-related concepts bias an agent to prefer navigating in the direction of the goal closet. However, in any given state, these concepts alone are insufficient to guide an agent because, alone, they do not take into account the position of walls or direction of the agent. Thus, the CC domain shows how concept-aware algorithms must in essence learn desirable actions from a conjunction of concepts, demonstrating that they can be used successfully in linear function approximation to produce a correct and optimal policy.

For CMAC, the additional object (the second block) adds cells within their tilings that have no knowledge transferred in, requiring them to learn that region of feature space. Similarly, CAFE forms new concepts to accommodate the presence of an additional block. The difference between the two featurization techniques, then, is how they cope with the burden of the anomaly's presence. In using only features derived from grounded state factors, the "unknown" regions of CMAC tiles share no structural similarity to the other features with which they coincide (as concepts do for CAFE).

For example, consider a CC-1 state in which the agent is just south of a red block in the center cell: the best action (recommended by learned weights) would be to push

the block north. However, now suppose the agent is later transferred to a CC-2 task, and encounters nearly the same setup, now with a yellow block just north of the red one. This yellow block occupies the doorway to the red closet, thereby preventing a push action from moving the red block. Even though the set of old tilings (from which the weights are transferred) is a subset of the new scenario's tilings, the knowledge they transfer over will negatively impact performance, biasing the CMAC agent into believing a useless action (going north) is best. This property is in direct contrast with CAFE, which gracefully accepts the addition of another block's new concepts in terms of those structurally and abstractly similar (its super-concepts). Using the previous example, the presence of the yellow block would be anomalous at the ground level, but through the FCA process, would also give rise to an abstract super-concept of a block (of any color or shape) being in the doorway to the red closet. CAFE would also produce an abstract concept of a block (again, of any type) being in the center cell: together, these abstract concepts help the agent circumvent the pitfall of useless behavior that would ensnare a CMAC agent. All together, this crucial difference in how feature-level knowledge is represented is substantiated practically by the results, showing the benefit of concept formation for transfer and anomaly reasoning.

5.4 Conclusion

In this Chapter I introduce a novel featurization technique, concept-aware feature extraction, that relies on the theory of formal concept analysis to assemble a hierarchy of feature clusters (formal concepts) that describe states of the world. I apply this technique to con-

textual bandits and reinforcement learning problems to show how it can reliably transfer knowledge abstractly to new, anomalous objects and situations.

For contextual bandits with concepts, I examine how the principles of anomaly reasoning can extend beyond classification to handle decision making under a variety of settings. With the Synthetic domain, which is comprised only of side information (where transfer is not needed), my CBC algorithm performs adequately, but not as well as the most straightforward approach. Upon introducing arm-specific features with the Mushroom domain (in which novel arms are anomalous objects, in this case mushrooms), I observe that concepts help to mitigate the negative effect of exploring new mushroom by interpreting them in terms of learned concepts. For the NetHack Monster domain, the agent is presented with a problem involving both side information and arm-specific features, where the agent must align its own label with a set of possible options (monsters). In this setting, one may observe the benefits of CBC most directly, such that a CBC algorithm spends an initial period of costly learning to eventually acquire a kernel of knowledge that results in policy that better minimizes long-term regret. The application of CAFE to CB problems exemplifies the type of anomaly interpretation and adaptation presented in the previous chapter.

For concept-aware reinforcement learning, I consider CAFE in combination with linear VFA in a transfer setting, where agents are trained on one task and deployed on another, more complex task. CARL algorithms operate by temporal difference, assigning weight to concept features based on how they influence the expected reward in future states. For the Traffic Light domain, I present a small MDP that can be solved easily by standard methods, while offering a pared-down challenge to examine how CARL algo-

rithms can disentangle concepts. Transfer to a harder TL task revealed how CAFE grants a transfer boost in terms of jumpstart and better asymptotic performance. I also examine CARL algorithms on the more complex Cleanup domain, the Cardinal Closets task simulating a robot that must move blocks of various shapes and sizes into the appropriate place. With CC, both learning and transfer are more reflective of real-world domains; as with TL and the results in CBC, one can see that the use of abstract concept features boosts the algorithms' ability to handle unfamiliar circumstances.

In the course of my experimental analysis of CAFE, I also examine its use in automatic feature clustering, its extensibility, and its facility for producing explainable and interpretable generalizations. Thus, overall I demonstrate how state abstraction, specifically by looking at how to form concepts from the environment, help us better generalize in terms of identifying, interpreting, and adapting to anomalies. In the subsequent chapters, I shift focus to more complex decision making that applies abstraction not only over states but actions as well, and I further consider how agents should generalize to new and uncertain scenarios in terms of abstract patterns of behavior.

Chapter 6: Hierarchical Reinforcement Learning and Planning

Just as we create concepts from our observations of the world, we similarly form habits from the natural repetition of useful behavioral patterns. Adaptability in terms of concepts requires anomaly reasoning; for habits, being an adaptable agent means generalizing these behaviors to new tasks and environments.

Consider an adaptable robotic agent that faces typical kitchen preparation tasks. Suppose the robot has prepared some basic recipes in the past, and we now want it to do something new: prepare an omelet. With concept formation, we might expect it to have learned concepts related to utensils, containers, and ingredients—the things that constitute its environment. Given a new kind of whisk and bowl, despite never having seen any of these particular designs, we would hope that it could interpret and handle them correctly.

However, in giving this kitchen robot a new recipe (to make an omelet), we instead tax its ability to adapt to new goals and generalize its previous behavior. The recipe may call for eggs to be beaten, specifying this step only with this high-level direction. What habits might the robot have acquired to help it adapt? If the robot has already prepared cakes, for example, we might expect it to have some familiarity with beating eggs. Ideally, it would know how to manipulate each new egg without dropping them, how to collect and use new tools (like the whisk and bowl), and how the eggs would appear when beaten

successfully.

Clearly this egg-beating subproblem of omelet preparation is a whole task unto itself, separate from the final goal of any one recipe. If the robot has already learned how to beat eggs in general, as a habit, by reusing this knowledge it may greatly simplify the overall challenge of adapting to the new recipe. In fact, it could treat each step of a recipe as a separate subproblem worthy of learning as a habit, explicitly representing them as skills that may be recalled. Then, the solution to any new recipe is a combination of habits over which it may reason as individual actions, learning new ones as needed. The difference is that, framing habits as high-level actions, they necessarily play out over a longer period of time, at a more abstract level. The mathematical theories and mechanisms needed for this type of temporally abstract decision making are what I introduce in this chapter.

6.1 Abstraction of Actions Over Time

In shifting the focus from concepts to habits, I propose an equally critical question for intelligent agents facing uncertainty: how should agents represent and reason about their behavior over time? Ideally, agents should plan how to reach their goal by learning and predicting the causal effects of their actions, as well as extrinsic changes in the environment, all while balancing requirements or constraints along the way. From this wide pool of concerns, the topic of reasoning over groups of actions (such as in a sequence or pattern) comprises a single but substantial issue. This is the problem of habit formation, or more aligned with terminology in literature, subtask learning and skill acquisition [81,82].

In this chapter I introduce the common solution: coalescing several discrete, single time-step actions into an aggregate abstract action that plays out over variable or multiple time-steps to achieve a deliberate and specific subgoal.

A Motivating Example

Suppose there is an intelligent agent tasked with piloting a taxi to collect and ferry passengers to their desired destinations. This overall task can be made arbitrarily complex, with any number of passengers and possible destinations, varying distances therein, and some cost on taking actions. The general form of a solution would be a route minimizing the distance (and thus time-steps) needed to reach the goal state, in which all passengers are where they wish to be.

In human decision making, our ability to reason abstractly over multiple timescales and objectives can been readily observed in communication with such an agent. What if, as a human observer, I wish to tell the taxi agent a solution as an action sequence, specifying each step, as opposed to explaining it in a natural language? With access to only primitive actions like moving one step in cardinal directions, the former ("go north, go north, go west, pick up the passenger, . . . ") quickly becomes tedious, verbose, and also over-fit to the most fine-grained particulars of the given task. In the latter case, I would be more comfortable using high-level directions ("go to the closest passenger, then drop them off, picking up any others along the way"). This declarative-imperative split underlies the idea of abstracting actions over time: as designers we care about stating *what* to do, not necessarily *how* to do it, and presume the agent will ideally find the best

way to achieve it.

Suppose we had an abstract action for moving and picking up passengers, and another for moving and dropping them off. Combined, reasoning over these would simplify plans by expressing behavior in a more general and natural manner. We would not need to care about the particulars of each navigational step, as long as the taxi gets where it's going efficiently, vastly reducing the raw number of states and actions that need to be considered to create viable plans. But how can our taxi agent *learn* to recognize useful patterns for abstract actions, and moreover *plan* over them? The answer is to reason with a *hierarchy* of abstract actions and related subgoals, applying a standard process of human decision making: when faced with a complex problem, break it down into smaller ones.

6.1.1 The Hierarchical Approach

The principle of *divide-and-conquer* rests at the heart of hierarchical decision making: the agent subdivides the task at hand into constituent parts, each a more focused and simpler *subtask* to solve, recursively repeating this procedure until attaining the most atomic decision problems: primitive single-step actions. Assembling a solution to the overall task begins by composing these atoms into low-level subtask solutions, composing these into ever higher-level subtask solutions, reusing any previously-solved subtasks without needing to recompute them. Crucially, a subtask defines a pattern that encapsulates multiple actions into a single component over which an agent may reason discretely. A given subtask, then, expresses a program of behavior that occurs over an arbitrary number of time-steps, not just one as in all previous discussions of decision making.

When planning over subtasks, an agent can reason abstractly: it limits the set of actions available, focuses on the most pertinent objects, and ignores aspects of the environment irrelevant to the task at hand. As the vehicle that ties abstract actions with subgoals, subtasks also facilitate the reuse of solutions: when encountering the same subtask again, such as a situation that requires reaching a subgoal visited once before, previously used plans and policies may be reapplied, thereby making complex problems more tractable. More subtly, a subtask is itself both a decision problem and, once solved, serves as a bundle of actions forming a self-contained policy or program that can be planned over and executed discretely. Ultimately, a hierarchical agent seeks a sequence of subtasks specifying optimal behavior, such as by maximizing future rewards or by taking the agent from its current location to a state satisfying all of its goals.

6.1.2 Top-down vs. Bottom-up

Over the past two decades of research in hierarchical methods applied to MDPs, the central concern under examination has been the representation of subtasks as temporally abstract actions. Across literature, two paradigms of hierarchical decision making dominate: the *bottom-up* learning and the *top-down* planning of subtasks.

The bottom-up approach most commonly arises in bootstrapped, temporal difference settings such as the options framework [150], MAXQ [34], and related techniques that operate using task hierarchies [18]. In this chapter, I explore these methods and discuss my contribution to the development of two ones: the expected-length model of options (planning via model-based RL), and portable option discovery (transfer learning

for RL).

The top-down view, in contrast, has greater roots in classical planning techniques, such as STRIPS [40] and its extension to abstract hierarchies [135], with a significant body of work originating from hierarchical task networks [39], including HPN task and motion planning [71] and goal task networks (GTNs) [6]. In the next chapter I delve more deeply into the background on top-down approaches, and I discuss my contribution to one in particular at length: planning with hierarchies of abstract Markov decision processes. Completing this discussion of hierarchical methods, the subsequent chapter articulates a novel, combined approach for bottom-up learning of structures that enable top-down planning, synthesizing and leveraging the benefits of both.

Altogether, I recapitulate hierarchical decision making as aiming (1) to **abstract** patterns of actions into subtasks that, when executed, occur over multiple time-steps; (2) to **decompose** hard tasks into subtasks; (3) to **transfer**, or reuse, solutions for subtasks; and, (4) to **synthesize** a combined bottom-up learning and top-down planning perspective.

6.2 The Options Framework

Perhaps forming the most foundational of hierarchical decision-making techniques, the options framework [12,96,150] represents abstract actions as *options* available to an agent under certain conditions. As a mathematical object, an option functions exactly like an action that takes multiple steps to execute. It differs from simple lists of actions, or macro-actions, in that it is reactive. More precisely, the option specifies (1) when it may begin, (2) when it is complete, (3) and how to behave under varying circumstances.

Definition 6.1 (Option). *An option ω is a* 3*-tuple* $\langle \mathcal{I}_\omega,\ \beta_\omega,\ \pi_\omega \rangle$ [150]*:*

1. $\mathcal{I}_\omega \subseteq \mathcal{S}$*, a set of valid initiating states;*
2. $\beta_\omega : \mathcal{S} \to [0, 1]$*, a termination probability distribution;*
3. $\pi_\omega : \mathcal{S} \times \mathcal{A} \to [0, 1]$*, a local probabilistic policy.*

The initiation set $\mathcal{I}_\omega$ are the states in which the option may begin to be executed; equivalently, an option may instead define a predicate $I_\omega : \mathcal{S} \to \{0, 1\}$ that denotes if the state satisfies some arbitrary initiation condition, effectively assessing membership in $\mathcal{I}_\omega$. When the agent initiates ω in state s_t, it successively follows π_ω up to some state s_{t+k} when the termination condition is met ($k \sim \beta_\omega$). In other words, each state s the agent enters when executing ω has a $\beta_\omega(s)$ probability of terminating the execution.

With these components, an option captures the notion of a subtask, expressing the idea of a single high-level action that achieves a particular subgoals in the MDP in which it operates. Assuming valid conditions ($\mathcal{I}_\omega$), executing the option equivalently executes a program (π_ω) of lower-level actions over variable time-steps that eventually attain some subgoal condition (β_ω).

6.2.1 Options & SMDPs

Given an option set Ω, extending an agent's possible actions (i.e., $\mathcal{A} \leftarrow \mathcal{A} \cup \Omega$) results in a *semi-Markov decision process* (SMDP) [150]. Under this formalism, the Markovian assumption is relaxed, to account for the variable time-steps an ω might take. In general, I consider an SMDP as defining a *flat hierarchy*, where its options have equal precedence as subtasks, referencing only primitive actions rather than other subtasks. Benefits of

planning over options in an SMDP include guiding exploration to useful subgoals and allowing temporal difference updates to a terminal state of ω to propagate across all value function approximations for connected states. Planning with options remains a popular topic in literature planning [101, 102, 139]. A drawback of this scheme is that an agent may invest more effort in planning with or learning options than would be needed in solving the base domain without them [68]. Thus, I will explore techniques that exploit their beneficial aspects while mitigated potential downsides.

6.2.2 The Multi-Time Model

RL on an SMDP usually requires learning a *model*, a representation of T and R to predict the causal effect of actions and options on future states and reward, with techniques called *model-based RL*. By collecting sample experiences to improve its approximation of environmental dynamics, a model-based RL agent then simulates future actions to find a way to its goal. Thus, model-based agents effectively *learn how to plan*. With options, this kind of agent needs a particular type of model that accounts for the semi-Markovian nature of variable time-steps updates and probabilistic termination. The solution is a modified Bellman equation, called the multi-time model (MTM), that estimates a transition and reward corresponding to the effects of each $\omega \in \Omega$ [150].

Definition 6.2 (Multi-Time Model). *For an MDP that includes options (i.e., an SMDP), the multi-time model* [126, 127] *comprises:*

$$T_\gamma(s' \mid s, \omega) := \sum_{k=0}^{\infty} \gamma^k \Pr(s_k = s', \beta_\omega(s_k) \mid s, \omega), \tag{6.1}$$

$$R_\gamma(s, \omega) := \mathop{\mathbb{E}}_{k \sim \beta_\omega, s_{1\ldots k}} \left[r_1 + \gamma r_2 \ldots + \gamma^{k-1} r_k \;\middle|\; s, \omega \right], \tag{6.2}$$

assuming γ given. Note the dependence of each equation on random variable k, expressing the time-step of stochastic termination.

For option models, SMDP or "intra-option" model-learning algorithms allow incremental Bellman-like updates to attain T_γ and R_γ, which can be computed while executing ω (or from replaying a sampled experience of ω). Overall, learning options is an off-policy, bottom-up process [150].

6.3 Key Questions for Learning Options

Effective options facilitate planning to reach time-distant goals while directing exploration towards useful regions of state space. However, three key questions face us when trying to generalize options as subtasks:

1. How can we **learn** option models more efficiently, abstractly?

2. How can we **transfer** learned options?

3. How can we **discover** options in the first place?

6.3.1 Learning Option Models

An option depends upon a π_ω that encompasses the entire state space. In attempting to learn a *model* for producing such a policy, the problem is compounded. In particular, the transition probability distribution may be complex to approximate for MDPs of any non-trivial size, while the reward function might take any arbitrary shape. Crucially, Equation 6.1 depends on a joint probability distribution over all possible values of k, which may become arbitrarily hard or inefficient to acquire or compute. This property means, for model-based RL agents, intra-option learning mixes the stochasticity inherent to the environment with stochasticity in both the option's policy and termination condition.

Existing work typically addresses these challenges in the algorithms themselves, such as the option-learning algorithm $Q(\beta)$ that enables faster convergence by learning β in an off-policy, where an agent is capable of modifying the termination condition of the option [59]. Similarly, *interrupting options* [100, 150] modify algorithms to let agents preempt option execution, re-planning to speed up learning. Instead of solely proposing a new algorithm, *compositional option models* [139] generalize the Bellman operator to work on recursive nesting of options, preserving the ability to incrementally learn models, and outlining a novel algorithm, option-option model iteration, to compute it. In Section 6.4, I follow this latter strategy of introducing a new theoretical definition of the option model itself. This formulation differs from these others by making option more sample-efficient to learn, retaining near-optimality while being learnable by any existing model-based approach.

6.3.2 Transferring Learned Options

Once an agent learn options, to make them more general subtasks, it would ideally be able to reuse them in MDPs of similar or variant high-level tasks. However, options are specific to the MDP upon which they are learned or originally defined. Transferring options to variant MDPs pulled from the same task universe may be impossible or infeasible, since they depend directly on states of their original MDP.

6.3.2.1 Transfer Options

Transfer Options (TOPs) [96] address this challenge. TOPs take options learned (or expert-defined) on one state space and transfer them to another. This process considers *source* domains (MDPs) and a *target* domain (MDP from the same universe) with different states and dynamics. TOPs collect contiguous state-action pairs from source policies, and re-apply their entirety to a new domain, given a mapping of source states to target states. Thus, TOPs make the critical assumption that an expert can specify a mapping of states from the source to the target. Together with initiation and termination conditions (the I_ω and β_ω are additional user input), creating a TOP ω takes the externally defined mapping and translates all state-action pairings (of π_omega) into the new domain; anything extraneous to the target domain from the source is simply ignored, while any state-actions not covered in the target must be learned. Hence, TOPs straightforwardly generalize knowledge with minimal exposure to negative transfer. What remains to be made are an intelligent creation of mappings, and the selection of partial policies to transfer.

6.3.3 Discovering Options

Option discovery covers a broad range of possible approaches. I focus here on learning pieces of policies that are composable into complete options.

6.3.3.1 PolicyBlocks

PolicyBlocks option discovery [121] extracts the components needed for options. Assuming source and target domains share a similar state space, PolicyBlocks identifies partial policies by creating a *merge* across those domains. Merging takes the union of state-action pairs in source policies, $\mathcal{S}_m$, joining them into a *partial policy* preserving consensus action across states while collapsing the mapping for unknown states:

$$\pi_m(s) = \begin{cases} \emptyset & \text{if } s \in \mathcal{S} - \mathcal{S}_m \text{ (no consensus),} \\ a & \text{otherwise (consensus action).} \end{cases} \tag{6.3}$$

Note that PolicyBlocks considers deterministic policies, functions of states to actions rather than state-action pairs to a probability. When solutions overlap over source MDPs, PolicyBlocks creates *option candidates* created by merging the various π_m, scoring them based on how many consensus states they contain and how many source policies concur with the non-zero state-action pairs. Beginning with the highest-scoring candidate, PolicyBlocks reifies it as an option while subtracting its consensus state-actions from all other candidates. However the drawback remains that source and target must share the initial state space; this problem inspires our approach in Section 6.5.

6.3.3.2 Related Methods

Other option discovery methods include a focus on identifying critical paths or landmark states [12, 24, 98, 104, 110, 145], clustering and graph partitioning [103, 141, 143], and more general "skill" discovery by chaining actions [81, 142]. Additionally option discovery in a transfer context has received some theoretical treatments in literature: analyzing sample complexity and outlining a "probably approximately correct" (PAC-SMDP) algorithm [23]; and reframing MDPs in terms of extrinsic state features versus intrinsic *agent-spaces* [80, 85]; and finding new error bounds by generalization of options as tasks with transition-based discounting [168].

6.3.4 Towards More General Options

Bringing these questions together, I offer two answers; (1) learning sample efficient models by abstracting over the expected time-length of options, (2) discovering abstract options that transfer to novel tasks.

6.4 The Expected-Length Model of Options

We introduce our novel contribution of the Expected-Length Model (ELM) for options. This section is based on joint work with fellow doctoral student David Abel, and co-authors Michael Littman and Marie desJardins [4]. My main contributions to this work are in the creation and design of ELM, the discussion of its differences with respect to the multi-time model, and the experimental results and analysis; the theoretical results related to ELM are simply summarized in this section, as David was primarily their author.

We contrast the expected-length model of options directly with the multi-time model. Consider Definition 6.2, and note that it operates over a potentially unbounded number of time-steps (k). Likewise, there is a related number of parameters with which to model the distribution. ELM differs here, by applying the principle of abstraction over the probability factored by time-steps. Specifically, ELM uses a point estimate instead of the full joint distribution. ELM takes the maximum likelihood estimation (MLE) of an option's time-steps; an agent using ELM options will assume this point estimate captures the underlying distribution to a sufficient degree.

To restate, ELM biases how option models are approximated. The resulting value functions learned over such options represent value with respect to *expected* time. In this work we will demonstrate that ELM leads to an empirical error without negatively effecting the rolled-out policies. Moreover, our theoretical work builds toward a more concrete understanding of tuning the degree of abstraction such that the overall model error is arbitrarily close to the MTM model. In other words, ELM generalizes over subtask transition details, but still provides optimal or near-optimal policies, even when composing ELM options into task hierarchies, layering subtasks of subtasks. What is saved are excessive requirements for samples and parameters (as with MTM).

Our core insight with ELM is that we need not model the full joint distribution of an option's possible outcomes, as MTM does. Instead, we simply estimate the expected length of an option rollout. More precisely, ELM explicitly models the transition and reward dynamics using the expected number of time-steps taken by an option, μ_k, which can be learned incrementally and stored trivially.

Definition 6.3 (Expected-Length Model). *For a given option ω, the expected-length model of options assumes the expected number of time-steps an option takes, μ_k, approximates the underlying probability distribution:*

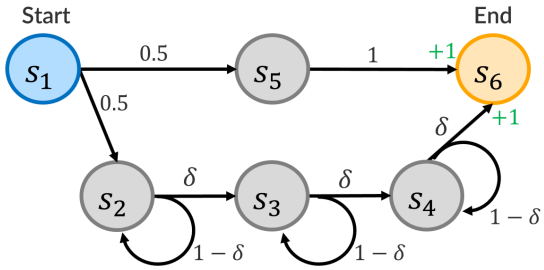
$$T_{\mu_k}(s' \mid s, \omega) := \gamma^{\mu_k} \Pr(s' \mid s, \omega), \tag{6.4}$$

$$R_{\mu_k}(s, \omega, s') := \gamma^{\mu_k} \mathbb{E}\left[r_1 + r_2 \ldots + r_k \mid s, \omega\right], \tag{6.5}$$

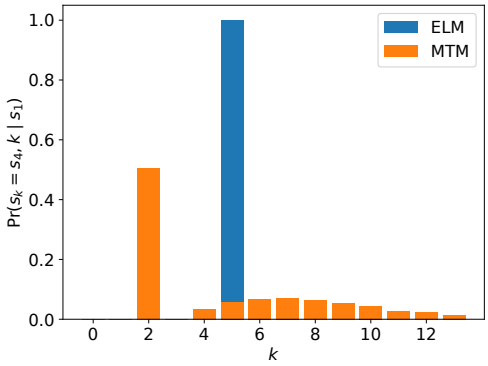
where $\Pr(s' \mid s, \omega)$ denotes the probability of terminating in s', given that the option was executed in s. Note that ELM no longer depends on a random variable, as MTM does (cf. Definition 6.2).

In contrast, the T_γ and R_γ of an MTM require computing and storing a joint probability of successor states, s', and k, the random variable whose observed event is dependent on the option's termination probability (β_ω). ELM inherently achieves a kind of true temporal abstraction: by only considering the *expected* number of time-steps an option may take, ELM ignores the fine-grained details and idiosyncrasies that manifest in the option's underlying dynamics. Thus, ELM captures a core notion of temporally abstract human decision making: in breaking down a hard task into simpler subtasks, at any given level we ball-park how long it may take to finish that subtask, and only when executing that subtask do we care about the particulars.

In the following subsections we articulate how ELM's unique form of abstraction is acceptable and desirable, leading to simpler models that still find (near-)optimal policies and improving performance in each experiment considered, all in fewer sampled experiences. Our methodology is tested in a variety of experiments, examining increasingly complex domains, and the effect of ELM in both flat and multi-level task hierarchies.



(a) A tiny, exemplary MDP. States are nodes, arcs are transitions labeled with their probabilities, using a parameterized slip probability δ, and non-zero reward is denoted in green.



(b) Model differences for $\delta = 0.4$.

Figure 6.1: An example of the differences between MTM and ELM for an option. In Figure 6.1a, consider the example six-state MDP. Further consider the option initiating in s_1 (blue) and terminating in s_6 (orange). Note that this option has two avenues, a guaranteed two-step path above and an arbitrarily long path below. This decision corresponds to the transition $T(s_2 \mid s_1, \cdot) = T(s_5 \mid s_1, \cdot) = 0.5$. Here is where ELM and MTM differ. Figure 6.1b shows the probabilities ELM and MTM place on the possible number of steps for the option, when $\delta = 0.4$. ELM represents only a point estimate: the expected number of time-steps taken by the option ($\mu_k = 5$). MTM, however, captures the full joint distribution for the transition, over all possible k time-steps taken by the option.

6.4.1 Intuition for ELM vs. MTM

By restricting our problem set to modified stochastic shortest path problems (SSPs), allowing goal and failure states, we show computing with ELM produces value functions that are within a bounded-error neighborhood of the optimal MTM value functions (tuneable by a sampling hyperparameter).

We present the six-state MDP in Figure 6.1a as a construction that accentuates the differences between ELM and MTM for the same option. Suppose an option goes from the blue node to the orange node, initiating in s_1 and terminating in s_6. For the sake of simplicity, let the probability of termination be $\beta(s_i) = 0$ for all $s_i \neq s_6$. We overlay the option's probabilistic policy onto the MDP, such that the arcs are assigned the direction

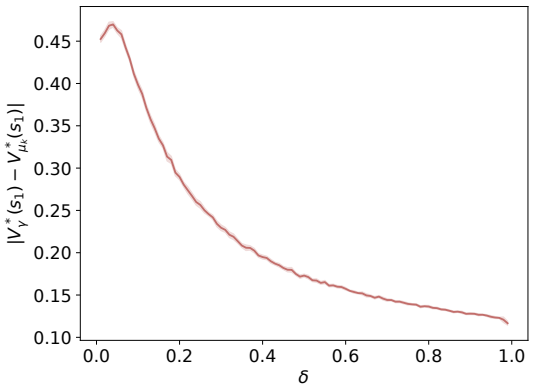


Figure 6.2: The difference in resulting value functions for options when varying the slip probability parameter $\delta \in [0.01 : 1.0]$ for the same option using MTM and ELM for the MDP defined in Figure 6.1. The 95% confidence intervals are visualized on the plot above, but are too negligible to be seen.

and probability that the option would follow. For example, the option policy from s_1 ends up in s_2 with probability $\frac{1}{2}$ and in s_5 with probability $\frac{1}{2}$.

The example includes a "slip" probability, denoted by the parameter δ, which we leverage to make the example arbitrarily more complex in terms of its stochastic behavior. This property can be best seen by the option's policy from s_2. With probability $1 - \delta$, the action will self-transition and the agent executing the option remains in s_2. With probability δ, the agent progresses into s_3, and the process repeats for s_3 and s_4. The alternate avenue offers a more deterministic two-step path from s_1 to s_5 to the terminal s_6.

Now consider the estimation of the transition into s_6 under MTM: $T_\gamma(s_6 \mid s_1, \omega)$. To construct a proper estimate, MTM must estimate the probability of termination in each state over all possible time-steps to determine $\Pr(s^{(1)} = s_6 \mid s_1, \omega), \Pr(s^{(2)} = s_6 \mid s_1, \omega), \ldots$. This computation involves estimation over arbitrarily many time-steps; in some cases, like this one, we might find a closed form based on convergence of the

geometric series, but agents cannot always intuit this fact from limited data. In contrast, ELM models this distribution according to μ_k, the average number of time-steps.

Given the true MDP transition function T, we run n rollouts of the option to termination. Supposing each rollout reports (s, ω, r, s', k), with r the cumulative reward received and k the number of time-steps taken, we can trivially estimate μ_k with the maximum likelihood estimator (MLE) $\hat{\mu}_k = \frac{1}{n}\sum_{i=1}^{n} k_i$. We can also estimate $\Pr(s' \mid s, \omega)$, the probability that ω terminates in s', by modeling it as a categorical distribution with $\ell = |\mathcal{S}|$ parameters. Then, we estimate each ℓ_i with an MLE.

To summarize: ELM estimates μ_k and $\Pr(s' \mid s, \omega)$, for each s' of relevance, by using an MLE based on data collected from rollouts of the option; MTM must estimate the probability of terminating in each state, at each time-step. It is unclear how to capture this infinite set of probabilities of value economically.

We visualize their differences in the quantity $\Pr(s_k = s_6 \mid s_1, \omega)$, for each k, in Figure 6.1b. MTM (in orange) distributes the transition probability across many step lengths k. Approximately half of the time, s_6 is reached in two steps via s_5; the rest of the probability mass is spread across higher values, reflecting longer paths (via s_2). ELM (in blue) instead assumes the option takes $\mu_k = 5$ steps. For both models, each non-zero bar represents a parameter that needs to be estimated, giving a sense of the difficulty in estimating each distribution. We also present the value difference under each model in Figure 6.2, which decreases to around 0.15 as δ tends to 1 (with $\text{VMAX} = 1.0$). This trend suggests that the higher the variance over expected number of time-steps, the more the ELM deviates from MTM.

In sum, this example highlights the following intuition: we need not decompose

future plans into the probabilities over all possible actions, over all possible time-steps; such reasoning can actually defeat the purpose of temporal abstraction.

6.4.2 The Difference in Learning Models

The goal of ELM is to simplify MTM to be able to estimate and compute the model of a given option more efficiently. MTM relies on modeling the outcome of a given option over all possible time-steps, which is impractical to compute even in small domains.

Estimation. Learning an option's MTM involves estimating infinitely many probability distributions. A reasonable approximation might involve limiting the sum to the first $\lambda = (1-\gamma)^{-1}$ steps as an artificial horizon, thereby yielding $\lambda|\mathcal{S}|^2$ parameters to estimate. In contrast, ELM requires learning the parameters of a categorical distribution indicating the probability of terminating in each state. With one multinomial for each state, any learning algorithm must estimate $2|\mathcal{S}|^2$ total parameters. Depending on the stochasticity inherent in the environment, option policy, and option-termination condition, estimating this smaller number of parameters is likely to be considerably easier ($\lambda \gg 2$).

Computation. The MTM requires performing the equivalent computation of a Bellman backup until the option is guaranteed to have terminated *just to compute the option's reward function* (Equation 6.2). Due to the decreasing relevance of future time-steps from γ, one might again only compute out to λ time-steps to determine R_γ and T_γ. Thus, computing R_γ is roughly as hard as computing the value function of the option's policy (at least out to λ time-steps), requiring computational hardness similar to that of an algorithm

like Value Iteration, which is known to be $O(|\mathcal{S}|^2|\mathcal{A}|)$ per iteration, with a rough convergence rate of $\tilde{O}(\lambda|T|)$ for $|T|$ as a measure of the complexity of the true transition function [94, 160]. Conversely, ELM is well suited to construction via Monte Carlo methods. Consider a single *simulated experience* $e = (s, \omega, r, s', t)$, of the initial state, the option, termination state, cumulative reward, and time taken. This experience contains each data point needed to compute the components of option ω's model (Equations 6.4 and 6.5), all sampled directly from the appropriate distributions. We highlight this property of ELM as desirable when the acquisition of samples is costly, as in robotics domains. With ELM, option models can be learned from these simulations, $\mathcal{E}$, with each $e \in \mathcal{E}$ needing only labels of where the option began, where it ended, how much reward it received, and how long it took. It is therefore sufficient to run a number of rollouts proportional to the desired accuracy when using ELM. Relying on such methods for computing MTM again requires estimating an arbitrarily large number of parameters, which is clearly untenable.

6.4.3 Theoretical Analysis

In addition to the experiments we present in this section, we conduct a theoretical analysis of ELM, with our main theorem bounding the value difference between ELM and MTM for Stochastic Shortest Paths (SSPs) with high probability under certain reasonable conditions. The theorems and related lemmas are proved in our conference paper to be published later this year [4]; David Abel is responsible for the majority of our progress on these theoretical results, and thus are described here only in summary.

We find that the difference between MTM (T_γ and R_γ) and ELM (T_{μ_k} and $R_\gamma\mu_k$)

may be bounded such that the value function learned using ELM results in near-optimal policies. Consider any option $\omega \in \Omega$, variance precision parameters $\tau > 1$ and $\delta = \frac{\sigma^2_{k,\omega}}{\tau^2}$, and all state pairs $(s,' s) \in \mathcal{S} \times \mathcal{S}$. Assuming options have non-zero probability of termination in every state, we determine with probability $1 - \delta$:

$$|T_\gamma(s' \mid s, \omega) - T_{\mu_k}(s' \mid s, \omega)| \leq \gamma^{\mu_{k,\omega} - \tau}(2\tau + 1)e^{-\beta_{\min}}. \tag{6.6}$$

Further assuming we restrict the case to only SSPs, we obtain it is always true that $|R_\gamma(s, \omega) - R_{\mu_k}(s, \omega)| = |T_\gamma(s_g \mid s, \omega) - T_{\mu_k}(s_g \mid s, \omega)|$. And, thus, with probability $1 - \delta$:

$$|R_\gamma(s, \omega) - R_{\mu_k}(s, \omega)| \leq \gamma^{\mu_{k,\omega} - \tau}(2\tau + 1)e^{\beta_{\min}}. \tag{6.7}$$

Moreover, we establish any policy over options π_ω with probability $1 - \delta$:

$$|V^{\pi_\omega}_\gamma(s) - V^{\pi_\omega}_{\mu_k}(s)| \leq \frac{\varepsilon(1 - \gamma^{\mu_k}) + \gamma^{\mu_k}\frac{\varepsilon}{2}\textbf{RMAX}}{(1 - \gamma^{\mu_k})(1 - \gamma^{\mu_k} + \frac{\varepsilon}{2}\gamma^{\mu_k})}, \tag{6.8}$$

where $\varepsilon = \gamma^{\mu_{k,\omega} - \tau}(2\tau + 1)e^{-\beta_{min}}$. As a consequence of these theoretical conclusions, we ascertain a loose though non-vacuous bound between MTM and ELM in the most general case; our subsequent experiments demonstrate, in practice, we observe much tighter bounds such that ELM efficiently and successfully approximates the multi-time option model.

6.4.4 ELM Experiments

We examine the utility of ELM as a means of learning option models under a variety of conditions, with tasks of increasing size and complexity. The main hypothesis we investigate is how ELM compares to MTM for learning and exploiting option models in SSPs.

6.4.4.1 Methodology

We frame each experiment as a hierarchical model-based reinforcement-learning problem. In this paradigm, an agent reasons with a collection of primitive actions and options, or a hierarchy of options. All models are initially unknown; or equivalently, the agent is only given an initiation predicate and termination probability, but no policy, $\langle I, \beta, \cdot \rangle$. Thus, the agent must estimate each option model through experience—we use R-MAX to guide learning [22]. R-MAX counts transition visitations and total rewards as they are observed. Crucially, unknown transitions are treated as providing maximum reward until they become "known" by being visited beyond some m threshold. It is here that MTM and ELM differ in application: a transition under MTM requires adding and updating as many parameters as needed across all k possible time steps, while a transition under ELM needs only update its running average, μ_k. Once a transition is known, its respective values in T and R are computed by R-MAX to be the observed totals divided by the state–action count. An option policy is then generated by running a planning algorithm with the R-MAX-approximated model; we use value iteration.

Our experiments each consists of 30 independent trials. Every trial, we sample a

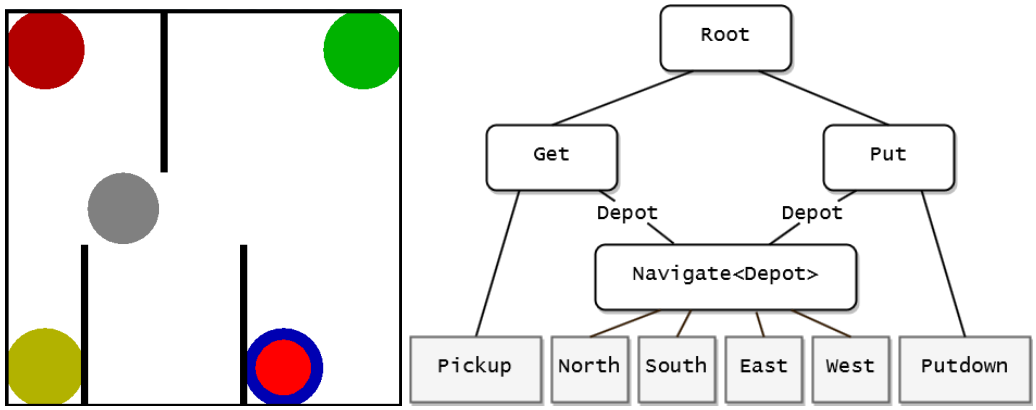
new MDP from the given domain (all MDPs in the same domain have identical transitions, states, and actions). Each MDP uses a goal-based reward function, providing the greatest reward at goal states, adhering to the properties of SSPs, and yielding the most negative reward at any failure states. A trial consists of 300 episodes, terminating at either a goal state or failure state or upon reaching a maximum number of steps. The task hierarchies are expert-defined and, for cited domains, are based on options or MAXQ task hierarchies in existing literature. We set $m = 5$ for the confidence parameter in R-MAX. Across all MDPs, $\gamma = 0.99$, and all transitions are stochastic with probability $4/5$ of an action "succeeding," otherwise transitioning with probability $1/5$ to a different adjacent state.

6.4.4.2 Domains

We experiment with the following domains: Four Rooms, Bridge Room, Taxi, and Playroom.

The **Four Rooms** domain [150] is a well-known gridworld with bottleneck "hallway" states between four larger, walled-off rooms, and a goal state at some random position. An agent may move north, south, east, or west, and possesses options for moving to the hallways, which may be initiated when the hallway is adjacent to the agent's current room.

The **Bridge Room** domain is a variant gridworld where a large central room contains a bridge of traversable cells that are flanked by "pits" (failure states). The agent starts on one side of the bridge, and the goal state is opposite, with both just outside of the interior room. Two corridors on either side of the central room offer safe but longer



(a) Example Taxi state. (b) PALM-ET, the classic expert Taxi hierarchy.

Figure 6.3: An example state and task hierarchy for the Taxi domain [34]. In 6.3a, objects are the taxi (gray), walls, depots, and passenger (smaller red circle, at the blue depot). In 6.3b, edge labels specify parameters passed from parent to child subtask (thus, for the state on the left, the hierarchy would specify four NAVIGATE options, one for each depot).

pathways. Differing from the Four Rooms domain, the agent is only given options for moving to the doorways between rooms. The bridge is short but crossing it is dangerous due to stochasticity. The ideal policy, then, is to use either corridor option around the bridge room.

The **Taxi** domain [34] is a classic hierarchical learning problem where the agent, a taxi, must collect passengers and ferry them to different destinations. We show an example state in 6.3a. Here, options are based on the standard MAXQ task hierarchy, visualized in Figure 6.3b: four NAVIGATE options (one each for moving between each destination depot, with all primitive movement actions); for each passenger, there's a GET option that can "pickup" (a primitive action) and a PUT option to "putdown" the passenger, with both GET and PUT able to use all NAVIGATE options; and, a ROOT option that can GET and PUT any passengers.

The discrete **Playroom** domain [82, 144] defines a complex, interlaced hierarchical planning problem. The agent has three effectors (an eye, a hand, and a marker) that must

be moved separately. The environment contains music and lights (both off) and several objects that can be interacted with if both the hand and eye are over them. There is a switch that turns the lights on or off, a green button that turns music on, a red button that turns music off, a ball that can be thrown towards the marker, a bell that rings when hit by the ball, and a monkey that cries only when the lights are off, the music is on, and the bell rings; the goal is to make the monkey cry. Playroom offers a tough challenge in that all three effectors must be coordinated and some work must be undone: buttons can only be pressed when the light is on, so any solution requires first turning the lights on, turning the music on, turning the lights back off, and throwing the ball at the bell. Following [82], our agent plans over the interact primitive action and options for moving each effector to each object.

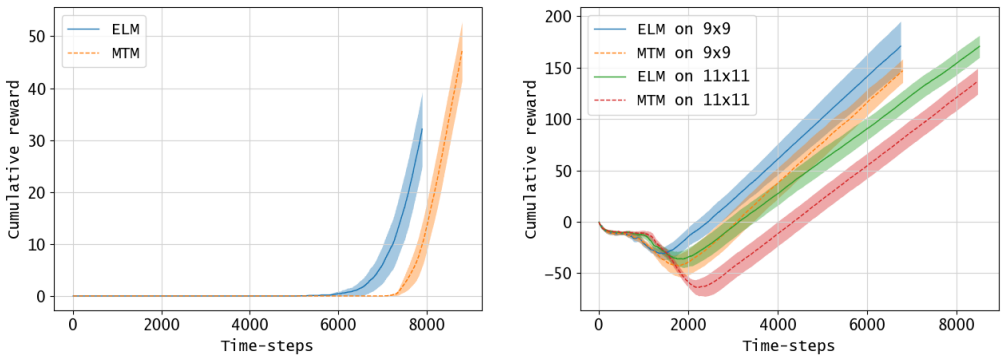
6.4.4.3 Results

We conduct experiments focusing on the speed and quality of learning ELM options models, in terms of discounted cumulative reward (performance) and time steps (sample complexity), compared to MTM. Figures 6.4 and 6.6 present performance curves with 95% confidence intervals for the domains that we discuss shortly in more detail. Overall, we observe that ELM and MTM attain the same asymptotic performance across every example, reflecting the fact that they both eventually converge to similar value policies for each task. Further, the results suggest that ELM often requires fewer absolute samples to achieve the same behavior.

In general, we find that, with all else being equal, ELM requires fewer samples

to reach near-optimal behavior. This fact is reflected by the graph of ELM terminating earlier than MTM when plotted over time steps in Figure 6.4a, given both are run for a consistent number of episodes. ELM more efficiently achieves the same trend. This result reveals how, under ELM, plans reaching the goal are formed earlier, how the agent more quickly finds a good policy. Consider the difference of the value functions learned under these models (Figure 6.5). The image displays the error that arises from the assumption ELM makes when planning over options, relative to MTM, while reflecting some noise due to stochasticity in the domain. However, upon inspection of this and all other trials, the overall shape of the value function for ELM and MTM is approximately the same. For example, in the trial from Figure 6.5, both $V^*_{\mu_k}(s)$ and $V^*_{\gamma}(s)$ ramp up in value towards the upper-right corner, from the three other corners. Most importantly, despite the difference in the value functions, the policies generated from both are identical; both MTM and ELM yield the optimal policy. The end result is that, while the option models learned under MTM are correct and optimal, those learned under ELM are near-optimal but acquired sooner, while still yielding the optimal policy.

We consider results on two variants of the Bridge Room domain, grids of size 9×9 and 11×11 (Figure 6.4b). The joints in the graphed curves reflect when option models solidify (the majority of transitions in R-MAX become "known") In the latter figure, as with Four Rooms, we remark that ELM begins converging earlier consistently, reflecting its ability to generalize more quickly about the expected length, and thus value, of the available options. In the former, however, the results are not statistically significant, and we see here a trade-off of ELM over MTM. For this smaller domain, the bridge is short enough that ELM may randomly happen to cross it safely several times. If this event



(a) Four Rooms.

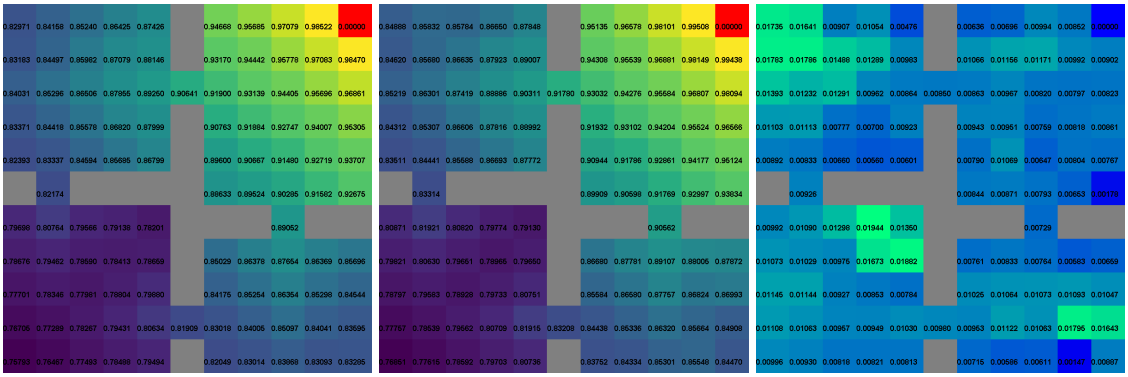
(b) Bridge Room.

Figure 6.4: Learning flat hierarchies of option models in relatively simple gridworlds. In 6.4a, ELM and MTM attain the optimal policy, with ELM dependably relying on fewer samples to begin solidifying its models. In 6.4b,

occurs, the agent learns to expect higher reward from the bridge option, negatively impacting ELM's overall performance until it eventually learns the impact of stochastically falling into a pit. Hence, the confidence interval of ELM on 9×9 in Figure 6.4b widens as ELM is less consistent across trials; we designed this domain precisely to exhibit this potential downside of ELM. Note that, while the ELM options here are not optimal and are subject to greater variance, the resultant policy converged to by the planning algorithm using these models *is* optimal.

For the Taxi domain, we consider the cumulative number of samples as task complexity increases from one to three passengers. For each, we discern that both learn models in relatively few episodes. In the case of one and two passengers (Figures 6.6a and 6.6b), the results are closely aligned, and the benefit of ELM over MTM is significant but minimal. For the largest task, three passengers (Figure 6.7), we observe similar results but draw attention to the lower variance among trials.

Figure 6.8 presents results, again measuring cumulative steps taken (so lower on



(a) Value under MTM, V_γ. (b) Value under ELM, V_{μ_k}. (c) The difference $|V_{\mu_k} - V_\gamma|$.

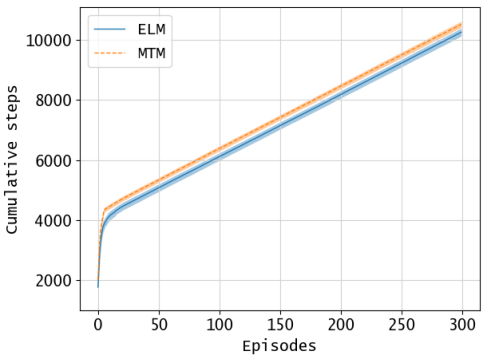
Figure 6.5: Visualizations of learned value functions in a Four Rooms task under (a) MTM, (b) ELM, and (c) their absolute difference. In this trial, the goal is in the upper-right corner, and learning occurred over 300 episodes. In (a) and (b), each cell contains the value of the state in which the agent occupies that position, from low (red/violet) to high (yellow). In (c), each cell instead reports the error, $|V^*_{\mu_k}(s) - V^*_\gamma(s)|$, visualized from low (blue) to high (green).

the y-axis means faster learning) in the discrete Playroom domain. Here, the patterns manifested in the other examples recur, though the two trends diverge later than in the Taxi experiments. This behavior is due to the immense state–action space that must be learned for the effector-moving options, such that, even as they are being learned, we see ELM's effect—favoring expected length leads to the generation of overall shorter plans.

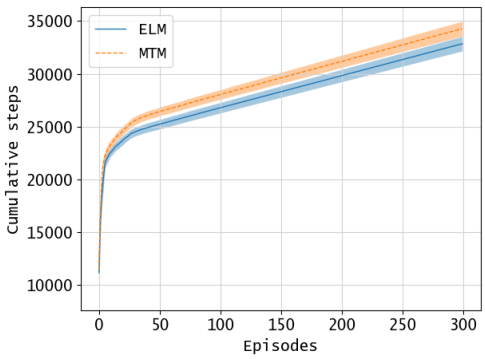
6.4.5 ELM Discussion & Context

With ELM, we propose an option model that is simpler to acquire with limited or no impact to performance, illuminating how it retains a reasonable approximation of MTM while removing the overhead in its construction.

In related work, learning models for use in making long horizon predictions has proven challenging. For instance, even ε-accurate one-step models are known to lead to an exponential increase in the error of n-step predictions as a function of the horizon [22,76],



(a) One passenger.



(b) Two passengers.

Figure 6.6: ELM experiments learning options for Taxi task hierarchies.

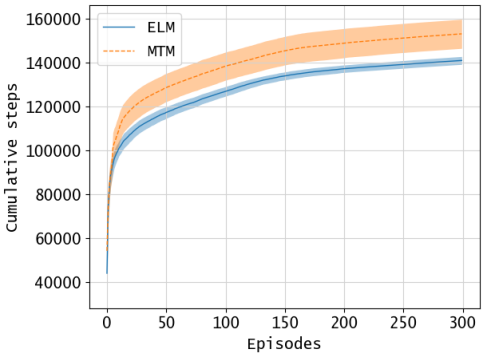


Figure 6.7: Taxi, three passengers.

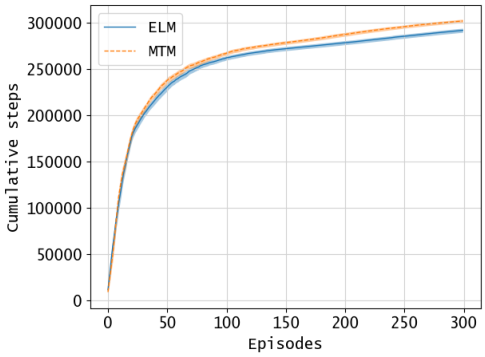


Figure 6.8: Playroom.

though recent approaches show how to diminish this error through smoothness assumptions [11]. Composing an accurate one-step model into an n-step model is known to give rise to predictions of states dissimilar to those seen during training of the model, leading to poor generalization [152]. Recent work has proposed methods for learning options that alter some aspect of the traditional formalism, either by treating option terminations as off policy [59], regularizing for longer-duration options [100], or composing option models together to be jointly optimized while planning [139]. It remains an open question, however, as to how to tractably obtain an option model.

In future work, we suspect that a nearby approximation of ELM can serve as a

sufficient replacement for MTM in richer classes of MDPs. Second, we foresee a connection between ELM and the problem of *option discovery*–we speculate that finding options with simple models may serve as a useful objective for learning. For instance, inherent stochasticity leads to higher ELM error. Thus, finding options that minimize this source of error may enable quick learning of options *and* their models. Finally, further analysis may shed light on the bias-variance trade-off induced by the ELM.

6.5 Portable Option Discovery

In the majority of previously discussed work, options themselves originate with a human designer. In these cases, the structure of the initiation and termination condition are pre-specified using expert knowledge. When options are discovered, such as with PolicyBlocks, they lack the ability to generalize to new tasks. Moreover, the number of options are also given beforehand, which can affect the overall performance and behavior an agent may learn [12].

In this section, we seek to describe an algorithm capable of discovering options that transfer among variant, related tasks. This goal poses a significant challenge as it requires composing disparately learned options while also generalizing them. Here, we present an approach that achieves just that, our research on Portable Option Discovery (POD). POD leverages a novel mapping and heuristic search approach for abstracting options that can be applied to existing algorithms to shore up where they are weak. For instance, POD can make PolicyBlocks partial policies transferable, and POD provide a means of discovering source policies that can transfer via TOPs. The content of this section is derived directly

from joint work, presented by Topin et al. (2015) [159], and I acknowledge the significant contributions of my colleagues and fellow students, Nicholay Topin, Nicholas Haltmeyer and Shawn Squire, the prior work of UMBC students Tenji Tembo, Michael Bishoff, and Rose Carignan [32], as well as the guidance of co-authors James MacGlashan and Marie desJardins. I supervised the design of the algorithms for POD presented here, with my primary contributions being in the writing, discussion of computational complexity, and analysis of experimental results.

6.5.1 POD Approach

We present three core aspects for discovering and assembling options that are portable:

1. a state abstraction function built on consensus of relevant objects;
2. a scoring mechanism for partial policies;
3. and a guided search over possible abstract policy mappings.

Throughout our discussion of POD, we assume an OO-MDP formulation of tasks, though the methodology could be generalized to any factored state space MDP. When we discuss "policies," we are referencing option policies specifically but omitting the word "option" for brevity's sake. Additionally we rely upon specialized terminology from the existing literature [32, 121]:

- *source policies*, a set of policies from one or more example task MDPs;
- *a merge*, the collected state-action pairs shared between two partial policies;
- and *option candidates*, the set of possible policies resulting from a merge.

From a high-level view, our option discovery focuses on converting from source policies to partial policies, extracting commonalities, creating abstract intermediate representations, and then grounding them to a novel task, yielding a set of specific options that carry over previously-learned behaviors. More specifically, we obtain an abstracted representation of subtasks, capturing knowledge of how an agent may act in a way conducive to generalization. POD accomplishes this goal by building abstract policies in option candidates suitable for grounding to novel tasks.

6.5.1.1 State Abstraction for Options

The first step is to apply state abstraction to map source policies into a common state space. Formally, suppose we have access to tasks pulled from the same universe or domain D, with a set of source policies $\mathit{\Pi} = \{\pi\}_D$. There also exists an implicit set of all possible states that could be expressed, $\mathcal{S}_D$, such as the range of states the classes of an OO-MDP could describe, and all primitive actions that are ever available, $\mathcal{A}_D$. Because each π may come from a different task MDP, each has its own $\mathcal{S}_\pi \subseteq \mathcal{S}_D$ states and $\mathcal{A}_\pi \subseteq \mathcal{A}_D$ actions. To generalize them, we require a state abstraction function $\phi : \mathcal{S}_D \rightarrow \tilde{\mathcal{S}}$.

Many possible ϕ may exist; we induce our ϕ directly by exploiting the nature of OO-MDPs. Recall that an OO-MDP possesses an object set $\mathcal{O}$ from which its $\mathcal{S}$ may be derived, and the abstract state space $\tilde{\mathcal{S}}$ would have its own set of abstract objects $\tilde{\mathcal{O}}$. In particular, we define the set of abstract objects $\tilde{\mathcal{O}}$ as a subset of the OO-MDP objects present across source domains, $\tilde{\mathcal{O}} \subseteq \mathcal{O}_\pi \; \forall \; \pi \in \mathit{\Pi}$. When there are multiple

objects of the same type in $\mathcal{O}_\pi$, each must be associated with some abstract object (with potentially many o mapping to one $\tilde{o}$). This process may be naïve, done straightforwardly but with limited efficacy. Crucially, our contribution includes describing how to enhance the creation of ϕ by considering the relevancy of ground objects to the source policy from which they originate. This approach identifies irrelevant objects in a $\mathcal{S}_\pi$, so they do not correspond to any abstract object in $\tilde{\mathcal{O}}$. Thus, unnecessary objects are eliminated, abstracted away.

When any $\mathcal{S}_\pi$ contains multiple objects of the same OO-MDP class C_i, multiple ϕ mappings to $\tilde{\mathcal{S}}$ exist. Given n_i objects of class C_i in $\mathcal{S}_\pi$, and $\tilde{n}_i$ objects of that class in the $\tilde{\mathcal{S}}_\pi$, there are $n_i!\binom{n_i}{\tilde{n}_i}$ possible mappings. If there are $|\tilde{\mathcal{C}}|$ object classes in the abstract domain, then the number of possible mappings is the product of these terms for each object class:

$$\Phi = \prod_{i=1}^{|\tilde{\mathcal{C}}|} n_i \binom{n_i}{\tilde{n}_i}. \tag{6.9}$$

A mapping scoring procedure allows us to alleviate this burden, such that heuristic search can discover a viable ϕ with certain desirable properties without the need to enumerate all of Φ.

6.5.1.2 Inverse Mapping for Consensus

To construct the abstract policy $\tilde{\pi}$ based on actions referenced across the ground policies, we define the inversion $\phi^{-1} : \tilde{\mathcal{S}} \rightarrow \{\bar{\mathcal{S}} \in \mathcal{S}_D \mid \phi(\bar{s}) \in \tilde{\mathcal{S}},\ \forall\ \bar{s} \in \bar{\mathcal{S}}\}$. To clarify, this one-to-many mapping projects abstract states to a *set of sets of states* that forward-map to the abstract state space. Thus, with $\tilde{\mathcal{S}}$ and ϕ^{-1} in hand, we can compute $\phi^{-1}(\tilde{s}) \rightarrow \bar{\mathcal{S}}$, the

set of source states to which any abstract state inverse-maps. Each $\bar{s}$ may recommend a different action across $\mathit{\Pi}$, so we must find the consensus action. This process is similar to the partial policies built by PolicyBlocks, but requires more complex steps to account for the inversion mapping. Assuming access to Q-values corresponding to the policies in $\mathit{\Pi}$ (which can be trivially determined if not readily available), we then acquire $\tilde{\pi}$ by finding the consensus actions across $\bar{\mathcal{S}}$:

$$\tilde{\pi}(\tilde{s}) = \arg\max_{a \in \mathcal{A}} \frac{1}{|\phi^{-1}(\tilde{s})|} \sum_{\bar{s} \in \phi^{-1}(\tilde{s})} Q(\bar{s}, a). \tag{6.10}$$

While we consider deterministic policies here, extending this method to handle consensus over stochastic ones could be done through interpolation, taking the average probability, or the median action.

6.5.1.3 Scoring Grounded Policies

Many possible policies may exist when grounding the abstract policy. Thus, we need a mechanism for judging how to best reconstruct a ground policy derived from the partial abstract policy while maximizing the overlap of recommended actions. We rely on a scoring mechanism to rank policies by assessing how much information they preserve when grounded relative to the original source policies. The score, then, comes from the amount of state-action pairs shared between the ground and abstract policies.

We define $\bar{\pi}$ as the grounding of $\tilde{\pi}$ to the original task MDP from which $\tilde{\pi}$ was derived, such that $\bar{\pi}(s) = \tilde{\pi}(\phi(s))$. To minimize reconstruction error relative to a source

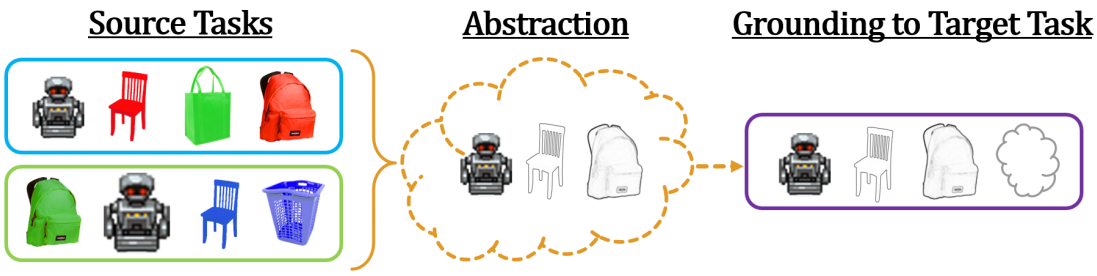


Figure 6.9: Conceptual diagram of abstraction and grounding. Objects are abstracted away from the source tasks and grounded to the target task. PPB performs a merge when grounding, while PTOPs does not.

policy π, we maximize a score defined as the overlap of consensus actions:

$$score = \frac{|\pi \cap \bar{\pi}|}{|\pi|}. \tag{6.11}$$

The diagram in Figure 6.9 represents how POD handles grounding.

6.5.1.4 Mapping Search

We now consider how to ground the abstract policy to a new target task MDP. Since we do not wish to score every possible mapping, we apply a step-wise greedy search (exhaustive search is computationally prohibitive).

First, we define the *greatest common generalization* (GCG) as the maximal subsets of objects appearing in both source and target. Formally, $GCG = \mathcal{O} \cap \tilde{\mathcal{O}}$ over classes, that is, regardless of the objects' instantiated attribute-values. We use the GCG in concert with Φ (Equation 6.9), with that set of mappings limited to just those resulting from merging pairs and triples of the source policies, and we determine b, its difference in number of objects from the GCG. In a step-wise manner, for each b, we consider each

Algorithm 2 PPB Power Merge

$n \leftarrow$ number of options
$\Pi \leftarrow$ set of source policies
$\Omega \leftarrow$ empty option set
$\omega, \tilde{\omega} \leftarrow null$
while $|\Pi| > 0$ and $|\Omega| < n$ **do**
  for all $\{\pi\} \leftarrow subsets(\Pi)$ **do** ▷ Get subsets, e.g., pairs and triples
    $g \leftarrow GCG(\{\pi\})$
    $\tilde{\pi} \leftarrow merge(abstract(g, \{\pi\}))$
    $\bar{\pi} \leftarrow ground(g, \tilde{\pi})$
    if $score(\bar{\pi}) > score(\omega)$ **then**
      $\tilde{\omega} \leftarrow \tilde{\pi}$
      $\omega \leftarrow \bar{\pi}$
  $subtract(\Pi, \tilde{\omega})$ ▷ Subtract states covered by $\tilde{\omega}$, as grounded to the $\mathcal{S}$ of each $\pi \in \Pi$
  $\Omega \leftarrow \Omega + \omega$
return Ω

mapping with one fewer object and take the one with the highest score: effectively, this process eliminates the least relevant object. In Section 6.5.2.1, we implement the above procedure as a greedy PolicyBlocks merge operation.

6.5.2 Two Portable Algorithms

Following the POD procedures discussed above, we derive two approaches based on existing option-related algorithms.

6.5.2.1 Portable PolicyBlocks

We now describe Portable PolicyBlocks (PPB), a novel variant of PolicyBlocks modified to discover portable options. Like PolicyBlocks, PPB relies upon merge and subtraction procedures. For PPB, we combine these processes into Power Merge (Algorithm 2). Power Merge takes raw policies from the same domain and yields the set of n option candidates. The core of the algorithm is a process that assembles as many of n as possible

given the policies. Starting from an initial subset of the source policies (in our case, pairs and triples), Power Merge first gets the GCG and produces an abstract policy. Crucially, the grounding of this abstract policy is scored relative any previously recorded one and kept only if it found to be the best candidate.

PPB Merge. We highlight the novel contribution of a modified *merge* operator and *subset* strategy. Specifically, we restrict the subsets to only those valid pairs and triples; upon empirical investigation, we observe that larger subsets beyond triples do not improve performance, as suggested by Pickett and Barto (2002) [121]. Our modified *merge* applies a greedy heuristic to merge step-by-step, and depends upon the aforementioned search procedure. For example, suppose we have four source domain policies π_1, π_2, π_3, and π_4 each containing a different k number of objects, with $k_{\pi_1} < k_{\pi_2} < k_{\pi_3} < k_{\pi_4}$. Then, merging proceeds in this order: π_4 is abstracted relative to π_3, $\tilde{\pi}_4$ and π_3 merge to form $\pi_{4,3}$, $\pi_{4,3}$ is abstracted relative to π_2, $\tilde{\pi}_{4,3}$ and π_2 merge to form $\pi_{4,3,2}$, and so on.

PPB Subtraction. The original PolicyBlocks subtraction procedure removes duplicate entries from all policies, and is not strictly applicable at the same point in PPB due to the latter's process of abstraction used in the creation of option candidates. More precisely, the policy subtraction operation replaces co-occurrence of the same state-action pair with $\emptyset$, a special "undefined" action for those states. In POD, source policies originate from MDPs of differing state spaces, making it ineffective to use subtraction directly (and subtraction at the abstract level is not appropriate). However, we may retain the abstract option candidate and ground it to each of the source policies in $\mathit{\Pi}$, and *then* apply

subtraction just as defined in PolicyBlocks.

PPB Grounding. We distinguish different cases of grounding, in particular separating the use of the *ground* operator in Power Merge and the later grounding of option candidates to target tasks. The former determines the grounded actions that would be preserved in $\tilde{\pi}$, relative to the GCG and initial policy, such that the resulting intermediary abstract policy $\bar{\pi}$ may be compared to the best known one using our scoring procedure. In the latter case, grounding is done online for option candidates applied to a task task. At test time, they must have their partial policies grounded to the specific states and actions of the given task. Here, the factored nature of OO-MDPs allows us to apply incremental grounding, circumventing the need to ever consider states beyond those specifically encountered by an agent during rollouts. Simply, as new states are explored, the option candidate's partial policy may encounter undefined state-action pairs. In such instances, "grounding" applies the mapping as previously discussed to find the known abstract states mapped to by the new, unseen state. Thus, any anomalous state is interpreted in terms of its nearest neighbors in the abstract policy space. Moreover, we leverage the nature of partial policies to scaffold the rest of the grounded options' structure: known states comprise those in $\mathcal{I}_\omega$ and unknown ones as having non-zero value for β_ω.

PPB Complexity

Computational complexity for PPB is dominated by the mapping procedure. In Algorithm 2, we consider complexity in terms of the process applied to each option candidate. That is, consider the partial policy to be incorporated into an option candidate. We must

compute the greatest common generalization by applying our abstraction process, performing a merge, and finally scoring (and grounding) the option candidate.

We begin with the GCG computation. Let n_{min} be the size of the GCG. Hence, we let n_{max} be the size of the *largest* input domain in any source or target policy being abstracted. Finally, as a convenience we define $\Delta = n_{max} - n_{min}$ to be the difference in number of objects, specifically the count of those that are abstracted away from the larger of the source or target domains.

In the computation of Algorithm 2, the process of abstraction must remove Δ objects. However, it considers up to n_{max} objects *every time* it is applied, in the worst case. Furthermore, the algorithm must iterate over every state in source policy in order to score the mapping properly. Thus, we obtain the complexity of a single step of the merge operation:

$$O(\Delta\, n_{max}\, |\mathcal{S}_\pi|). \tag{6.12}$$

The scoring procedure also adds a term to the overall complexity of PPB. Scoring is applied to the resulting option candidate from the merged source policies. Importantly, we note that scoring (in PPB) requires the option candidate to be grounded to each of the source policies, because the score is based on *all* source states that contain the same state-action pairs as the candidate. The process of grounding used here in scoring relies on the same complexity as in the aforementioned merge. Finally, as with standard PolicyBlocks, the state-action pairs covered by the candidate are subtracted from the source policies. That is, given the highest-scoring candidate, we must again iterate over the source policies. The complexity in subtraction is therefore dominated by that of the scoring step.

Taken all together, the overall complexity of PPB, with the additional terms are derived from our use of PolicyBlocks, arrives at:

$$O\left(k\,|\Pi|^3 \sum_{\pi \in \Pi} (\Delta\, n_{max}\, |\mathcal{S}_\pi|)\right). \tag{6.13}$$

Most crucially, it is practically infeasible to use the power set of source policies when creating option candidates, a drawback inherited from PolicyBlocks. Therefore, some heuristic must be applied to restrict or guide the selection of a subset. Following Pickett and Barto's suggestion in the original PolicyBlocks definition [121], we consider only the set of all pairs and triples (and not the full power set), yielding $O(|\Pi|^3)$. Given these, each combination of policies need only be examined k times, for each k option being created. We also note that, in practice, we observe that PPB scales much better than this reported worst case setting. This empirical behavior is, in part, owed to the fact that the set of source policies will diminish with each iteration of PPB.

6.5.2.2 Portable Transfer Options

We implement POD directly into TOPs and call the result Portable Transfer Options (PTOPs). As discussed in Section 6.3.2.1, a central drawback of TOPs lacks an informed, automatic way to create mappings between source and target states. POD interfaces with TOPs to achieve just that: the combined approach of abstraction, scoring, and mapping search serve to define the translation of TOPs from one state space to another. The only remaining issue is to select the best among option candidates; because there are no good heuristics for *a priori* determining the performance of PTOPs in the target task, we simply

transfer all option candidates. For PTOPs, the online grounding of option candidates to undefined regions of its partial policy follows the same strategy as in PPB.

PTOPs Complexity

We report the complexity of POD applied to TOPs. First, we note that the point of POD here is to provide an autonomously-constructed mapping for the transferred options (avoiding the need for an expert). In this case, such mappings require computing the size of the GCG, n_min. Thus, we obtain $O(n_{min})$, since the source policies are independent of each other, and only n_min objects need be considered.

We reuse the variables n_max and Δ as discussed in the PPB Complexity section. Consider the mapping search process following a greedy heuristic. The abstraction step has will iterate at most Δ times, one step for each object by which the source and target differ. In the worst case, the iteration must consider up to n_{max} objects for abstraction. Because we score as we go, this step further entails iterating over all states, giving us the term $|S_\pi|$.

Taken together, the GCG computation is dominated by mapping search. The resulting worst case complexity for PTOPs is thus:

$$O\left(\sum_{\pi \in \Pi}(n_{min} + \Delta\, n_{max}\, |\mathcal{S}_\pi|)\right), \tag{6.14}$$

or equivalently, since the GCG term is clearly dominated by the abstraction and scoring

term, the complexity is:

$$O\left(\sum_{\pi \in \Pi} \Delta\, n_{max}\, |\mathcal{S}_\pi|\right). \tag{6.15}$$

6.5.3 POD Experiments

I now discuss the experimental methodology we use in our implementation of POD. The goal with this investigation is to assess the two major aspects developed in POD, the abstraction mapping process and the scoring mechanism. We therefore concentrate our experiments on options in two domains comparing transfer under different conditions, using several different state space sizes to demonstrate POD's generalizability, measuring performance using the average cumulative reward over a series of learning episodes.

Methodology & Setup. I refer readers to the original paper for a more extensive discussion of our empirical methodology [159]. In summary, our experiments each use 20 trials (e.g., every trial applies a different seed for pseudorandom number generation). Every trial follows a complete transfer process, from training on a set of 20 randomized source tasks to a randomized target task evaluation, in a domain with a differing layout or number of objects. Our results report confidence intervals ($p < 0.05$). We maintain a standard learning algorithm across each of the transfer techniques; in all experiments, we use intra-option ϵ-greedy QL ($\epsilon = 0.025$) [150]. Other experimental hyperparameters include the probability of termination (for each ω option, a flat probability β_ω of 0.025, with 1.0 in goal and failure terminal states), a pessimistic initialization Q-value initialization, and a consistent -1 per-step reward.

Transfer Details. The main transfer metrics we consider are asymptotic performance of the algorithms applied to the target. Thus, in assessing the graphs we generate, focus should be placed on the curve or trend of each algorithm, noting any jumpstart in the initial episode, as well as the point of convergence, at which the algorithm has settled on a policy. The experiments are gauged against an oracle, the algorithm using an option with the "Perfect" policy. "Perfect" here provides an heuristic for the theoretical yet impractical-to-achieve (without expert knowledge) upper bound on performance. In other words, the optimal solution for the target task. Additionally, the transfer methods differ for the PolicyBlocks and TOPs approaches. The TOPs methods transfer all source policies, while PPB and RPB only apply the top-scoring single option candidate. The original paper discusses the transfer details in greater specificity [159].

Algorithms Tested. Our methods are Portable PolicyBlocks (PPB) and Portable TOPs (PTOPs). As previously discussed, the existing techniques lack either inherent transfer and require expert intervention to be reapplied in new target tasks with different numbers of objects. Thus, in order to more fairly compare our methods with the original ones, we include results for PolicyBlocks and TOPs with transfer via *random* mappings, both in the abstraction and the grounding procedures. These algorithms, RPB and RTOPs, respectively, highlight what naïve transfer would look like, showing the value added by the specific scoring heuristic we outline with POD.

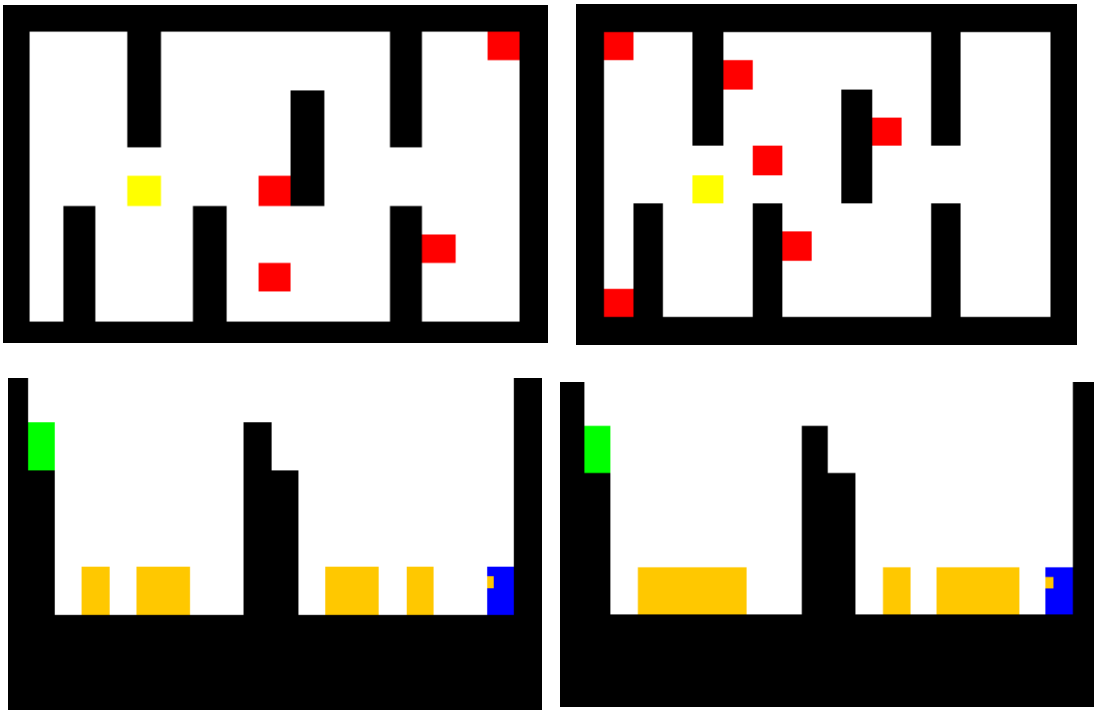


Figure 6.10: Example states: in Block Taxi (top row), the agent is yellow, passengers are red, and walls are black; in Block Dude (bottom row), the agent is blue, blocks are yellow, and the exit is green.

6.5.3.1 Task Domains and Results

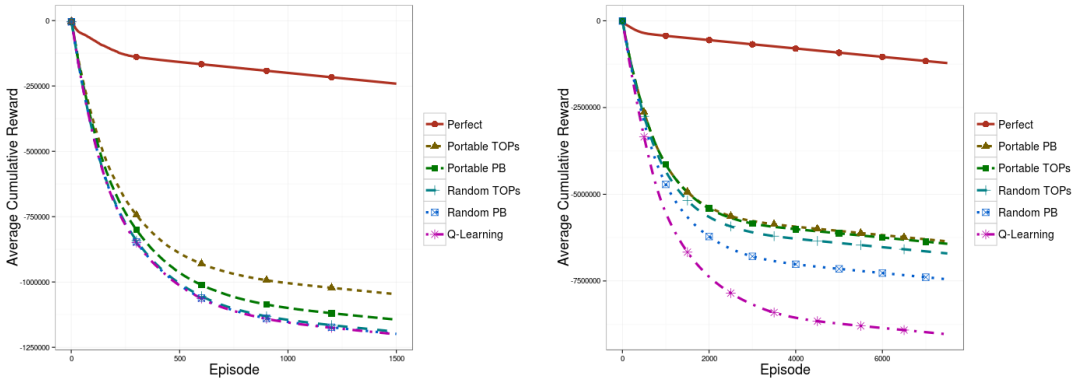
Block Taxi Results

The Block Taxi domain is related to Taxi [34], but consists solely of passengers and block walls (that occupy a cell rather than gaps between them). Rather than traveling between multiple depots, the agent's goal is to transport every passenger onto a specified location. The taxi agent may drop off a passenger at any space, adding to the challenge. For Block Taxi, we transfer from tasks with many-to-few objects, from few-to-many, and also highlight a more extreme case of few-to-many.

First consider the case of transferring from harder tasks to easier ones, with fewer objects and thus fewer steps to complete. Results for this scenario are shown in Fig-

ure 6.11a. We interpret the graphs by gauging how much cumulative reward the algorithm can accumulate over episodes (and thus, higher is better). Here PTOPs achieves a statistically significant improvement over the other methods, both in terms of raw reward, but also in the rate to which it reaches a good quality policy (reflected in the curve's time to convergence). The other algorithms overlap in their confidence intervals, so drawing a conclusion about their usage in this case (many-to-few) is limited. Randomized mappings do not hinder performance (we see no negative transfer relative to QL), but their transferred knowledge is unhelpful without some mechanism for better leveraging it, like our scoring procedure in PTOPs.

Transferring from the few-to-many (4 to 6 passenger) tasks, we observe a more stark difference (in Figure 6.11b). Noticeably, all methods improve a great deal over QL. The POD methods perform best, though not statistically significantly better than RTOPs. Our initial belief was that this lack of difference could be explained as follows: training on the smaller tasks lends less behavioral knowledge to the learned policies once transferred to to *larger* tasks. Essentially, there is an upper limit to how much behavioral benefit can be extrapolated, such that transfer is still important but making scoring less useful over simply applying randomized mappings (at least, as seen in RTOPs). However, we discount this theory in the next experiment. Consider the case of an even more extreme transfer, from two to seven passengers, with results shown in Figure 6.12a. Here, the difference between scored POD methods and randomized mappings *is* significant; PTOPs has the best performance, with PPB second. Both achieve higher reward and reach the optimal policy faster than either RTOPs or RPB. Thus, the advantage of scoring for POD may be hard to predict from task complexity alone, and warrants further investigation. We



(a) Block Taxi, 6 to 4.

(b) Block Taxi, 4 to 6.

Figure 6.11: Performance in the Block Taxi domain. In Figure 6.11a, transfer is done from tasks of six passengers to four passengers. These results highlight the value of scoring, since the methods with a random mapping degrade to baseline performance. In Figure 6.11b, transfer is done from four to six passengers. In this case, transferring to a harder task, there is a clear and significant benefit afforded by discovering and using portable options.

also note that RTOPs and RPB also improve significantly over the baseline, confirming that randomized mappings can give positive transfer.

Block Dude Results

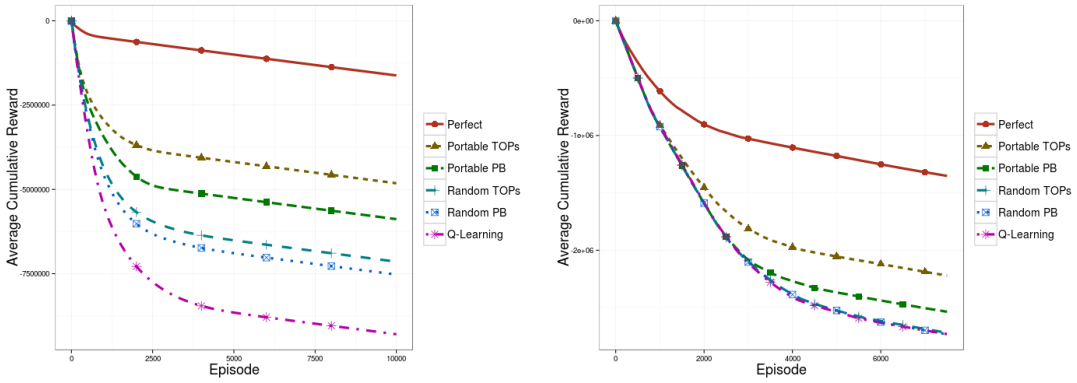
The Block Dude tasks are based on the game of the same name.[1] We treat Block Dude as a typical gridworld, but viewed as a "side-scroller" rather than top-down. Each task consists of a level from which the agent must escape by reaching a door. Solid walls block its path, and thus the agent must collect and stack blocks to surmount the walls and reach the door. Level layouts are randomized for reach trial. The agent occupies a cell (x and y coordinates) and includes a direction (left or right) in which it faces. The agent may move forward, move backward, pick up blocks, and drop them. If the agent is not carrying a block, the agent may collect one if it is adjacent to one and facing it.

[1] http://azich.org/blockdude/

Gravity applies, causing the agent and blocks to fall down to the lowest open cell. The agent may climb as well, moving up and forward if there is a block in front of it under an open space in the direction it is facing. Again, the goal is to stack blocks, often into staircase-like structures, making a path to an elevated or otherwise hard-to-reach door (exit), terminating the episode. Importantly, the agent may reach a state from which it cannot recover, so episodes end after a certain maximum number of steps.

Figure 6.12b shows the results in Block Dude. Here, the randomized mappings offer no improvement over the baseline QL. RPB and RTOPs see no benefit because the variety in Block Dude makes most mappings unhelpful (and thus, a random selection of them adds no benefit). However, both PTOPs and PPB achieve statistically significantly better results. Upon investigation of the success of PTOPs over PPB, we found that noise in the source policies was causing interference. In particular, PPB's merge procedure abstracted away more details than PTOPs, including useful edge case state-action pairs. We find retaining information while scoring is the most consistently useful transfer strategy. Thus, we content that, in general, as the number of possible mappings increases, the utility of our scoring mechanism for POD also increases.

Overall, from all experiments we consider, the combination of abstraction and scoring (e.g., heuristic search over mappings) provides a benefit in most cases. Though the scoring method never negatively impacts performance (and therefore there is no reason *not* to use it), we infer that it does not always yield the best possible mapping. Hence, it remains an open question for future work to find alternative mechanisms for scoring or more principled approaches to selecting policy subsets prior to merging.



(a) Block Taxi, 2 to 7.

(b) Block Dude (random layouts).

Figure 6.12: Performance on Block Taxi and Block Dude. In Figure 6.12a, transfer is done from tasks of two passengers to those of seven passengers, shedding light on the immense value of portable options when task complexity differs greatly. In Figure 6.12b, the task is difficult to learn even for the oracle option upper-bound. Note that both PTOPs and PPB begin converging more quickly with improved asymptotic performance relative to the random and baseline methods.

6.5.4 POD Summary & Context

In summation, the POD framework offers a comprehensive framework for performing automated, bottom-up, option discovery that results in significant knowledge transfer across tasks of varying complexity in OO-MDP domains. We describe the Portable PolicyBlocks (PPB) and Portable Transfer Options (PTOPs) methods. We assess their performance relative to a baseline and an oracle. Crucially, we analyze a heuristic mapping and scoring mechanism that, when incorporated into POD methods, allows them to select reasonable and valid mappings from the combinatorial set of possible mappings, outperforming the same methods with random mappings.

We draw a connection between POD and a wider literature of option discovery and the less-explored topic of option transfer. Notably, POD provides a complete procedure from learning in the source domains to the autonomous mapping of options abstractly

into a target domain. What is gained here, over related work, is the elimination of expert knowledge. Note, though, that we also make an assumption for POD to work: that the domain is defined in terms of objects. Previous efforts in option discovery require direct analysis of the state space: local graph partitioning [143], clustering [103], bottleneck state discovery [104, 110], and betweenness centrality [141]. POD removes the assumption of an expert or the requirement to perform *a priori* analysis. POD more reasonably requires an OO-MDP specification, no analysis or direct access to the state space itself. In many instances, this requirement is trivial since any factored state space MDP can put its state factors (features) in arbitrary boxes, calling those objects. This property (eliminating expert knowledge) makes POD ideal for cases where the full MDP is not known beforehand, which is common in many realistic domains, or where the reward functions differ among source and target; similarly, POD's autonomy may lend better to application on partially-observable MDPs, common to robotics settings, which we hope future work will explore. In terms of option transfer, one important method introduces a factoring across what is intrinsic and extrinsic to the agent, what may be called the agent-space and problem-space, respectively. In keeping the state spaces of options factored among an agent-space and problem-space, an agent's learning can occur at two levels, so that one may be varied while the other is held constant [80, 85]. Using an agent-space allows options to be learned and transferred to new problem-space tasks; however, the work cited here relies on some degree of feature engineering, such defining useful relational attributes (e.g., a feature that captures "the nearest block" to help the agent learn in its agent-space). Our hope with POD is that we can bring such ideas in option transfer into an option discovery setting, alleviating the burden of expert knowledge by performing

abstraction algorithmically.

6.6 Conclusion

This chapter has introduced the central paradigms of hierarchical RL and planning, outlining the difference between top-down and bottom-up approaches to handling subtasks. Taking the latter as a starting point, I cover the well-studied Options formulation of subtasks, and identify key questions relating to learning them. In terms of acquiring option dynamics efficiently, without sacrificing performance, I introduce the expected-length model of options (ELM) and demonstrate its desirable properties over the multi-time model in a domains of increasing complexity. The disparate goals of learning to create options and then transferring learned options are united in our portable option discovery (POD) framework, for which we demonstrate transfer under a variety of circumstances, and analyze the importance of heuristic search and scoring mechanisms for finding abstractions that generalize and improve transfer. Altogether, the bottom-up approach to subtasks creates practical representations derived directly from the source environments. While much of prior research results in options that are tightly coupled to their source, I show how such methods can be made to transfer knowledge, as exemplified by the reward and transition approximation of ELM and the generalized object-based state abstractions of POD mappings. In the next chapter, I investigate a different, top-down approach within a planning context, and investigate how making a new structural assumption about subtasks breaks their close ties to the source domain and greatly improves the ability to handle temporal abstraction at multiple levels.

Chapter 7: Abstract Decision Hierarchies

7.1 Introduction

I have explored how hierarchical decision making may be achieved through a bottom-up strategy, where agents learn to create and solve subtasks by assembling the components of an option. Now I consider the inverse, top-down paradigm where an explicit hierarchy is imposed on an agent's understanding. Critically, this approach invariably relies on a structure, the "hierarchy," that encodes some form of domain knowledge, often in the form of subtask-dependent admissible actions and state abstraction (including arbitrary, nonlinear mappings not based on state aggregation). Such knowledge most often comes from human experts, as is the case in the examples discussed in this chapter; however, the next chapter will explore methods for agents to learn both subtasks and hierarchies.

My focus also shifts from *learning* to *planning*, assuming that we can provide models for subtasks, while seeking to have our agents figure out the solution to long-term, complex tasks. In the top-down paradigm, by assuming some prior knowledge we gain robustness to the problems challenging the algorithms in the previous chapter. Namely, it helps address the issue of combinatorially expanding ways of considering objects and actions as their numbers increase. Introducing abstraction to hierarchical decision making means that each subtask need only consider the objects and actions most relevant to its

individual task. In particular, this chapter covers the background on top-down approaches including MAXQ (for planning) and *abstract Markov decision process* (AMDP) hierarchies, with material based upon discussions in Gopalan et al. [55] and Winder et al. [171].

7.1.1 Motivating Example

Consider a "taxi" scenario, in which the agent (a taxi) must ferry passengers among various stations. The optimal policy minimizes the overall distance it takes for the taxi to get to each passenger and put them each at their desired location. Even in describing the task in natural language, a clear hierarchical decomposition is expressed: an agent must both "get" and "put" passengers, with both of those steps requiring the agent to "navigate" its environment. Recognizing this breakdown is exactly what was achieved by Dietterich (2000) [34] in the original presentation of the Taxi problem, culminating in the hierarchy shown in Figure 8.2b. This *task hierarchy* represents subtasks as nodes in a directed graph from the ROOT task down to leaves, primitive actions of the ground MDP, and arcs denote that a child node provides an action available to the parent subtask.

In Taxi, ROOT captures the overall goal, while GET is parameterized over passengers and satisfied after an agent performs NAVIGATE and "Pickup" of the selected passenger. Likewise, a parallel recursive completion of subtasks down the other side of the graph is needed to PUT the passenger. Clearly, restricting the set of available actions at any time directly reduces an agent's search space. In addition, state abstraction may eliminate objects irrelevant to the task, such as walls for the *Get* and *Put* tasks, or use high-level relations to express objects' locations more abstractly. For example, one could

be "at(passenger, location)" for the *Root* subtask, converting from grid positions (e.g., x-y coordinates) to re-consider states in terms of relations or predicates. In sum, an abstract task hierarchy such as this one for the Taxi domain reflects how human designers frame and communicate hierarchical task descriptions in terms of subgoals, facilitating the encoding of knowledge. The ease of this goal-focused top-down approach for expressing subtasks abstractly may be compared with that of options, where providing a full policy for each option is not as straightforward process for human designers. Moreover, capturing repeated patterns of behavior in subtask policies allows them to be shared among all parent subtasks, facilitating reuse and efficiency.

7.1.2 Subtasks in a Hierarchy

For an agent planning top-down, a hierarchy of subtasks enhances its ability to consider temporally-distant states and efficiently reach more significant points of interest. Chapter 6 examined task hierarchies for reinforcement learning with options; here, I reframe them in terms of planning and take a closer look at MAXQ in this model-based context, as well as task hierarchies of AMDPs.

7.1.2.1 MAXQ

MAXQ [34] achieves temporal abstraction via task hierarchies. A MAXQ agent reframes a base MDP in terms of smaller MDPs that work together to construct a piece-wise value function for the overall task. Formally, given an MDP M, MAXQ decomposes M into a set of n MDPs $\{M_0, M_1, \ldots, M_n\}$, one for each subtask. Computing the value function of

a policy proceeds by replacing the recursive part of the standard Bellman equation, whenever the policy would lead to a child subtask, with the addition of that child's *completion function*, the core mechanism for MAXQ value decomposition.

MAXQ uses the idea of a completion function to represent the expected discounted reward of current task i after completing subtask (or action) a while in state s, $C(i, s, a)$. Then, action-value is decomposed recursively, $Q(i, s, a) = V(a, s) + C(i, s, a)$, in terms of the value of the state for the child a (which depends on Q at that lower level) plus the completion function of the respective task. Thus, planning with MAXQ (such as in [35]) is an inherently *bottom-up* process; I will compare this to top-down planning using AMDPs.

MAXQ achieves a recursively optimal solution, where the policy is optimal at each level of the hierarchy. Options differ in providing a hierarchically optimal solution that achieves the maximum reward at the base level [34, 150]. Therefore, plans in MAXQ can be suboptimal from a global perspective. However, in trading total optimality for near-optimal solutions, recursively optimal methods can offer significantly faster planning times.

7.1.3 Top-Down Approaches

In a bottom-up process such as options or MAXQ, planning requires expansion of each node recursively down the hierarchy to determine value or action-value. Top-down approaches attempt to avoid this taxing search by focusing only on the child subtasks worth expanding. The relevant history of top-down search begins with STRIPS [40] and its

hierarchical extension *abstraction-based* STRIPS (ABSTRIPS) [135]. As a means-ends analysis planner, the STRIPS algorithm performed problem-solving by searching ahead (hence, planning) to a goal state through possible states following from some initial state, given a set of conditions and operators. This propositional "action language" approach may be viewed analogously to the stochastic framing of MDP transition dynamics, especially for factored structures like OO-MDPs where the changes in state factors for each action are modeled probabilistically via dynamic Bayesian networks. Motivating ABSTRIPS, the central drawback of STRIPS is its limited planning horizon for a "reasonably complex domain" that inevitably fell into a "combinatorial quagmire" [135]. By first identifying abstractions based on *criticalities* of action preconditions, ABSTRIPS constructs abstract plans in descending precedence of difficulty [79]. Thus, it effectively decomposes the search space of a problem into subtasks arranged in a hierarchy of complexity. The net effect of ABSTRIPS over STRIPS is substantially reducing the number of branches that must be explored while searching.

Another iteration of top-down planning bears the name of hierarchical task networks (HTNs). This family of related algorithms builds upon a STRIPS-like approach, typically from a partial-order planning setting. Examples include NOAH, NONLIN, UMCP, and SHOP [39]. Planning with HTNs typically requires the task network itself (as with a task hierarchy, specifying the relations among primitive and non-primitive subtasks), the primitive operators, and the non-primitive "methods" (high-level operators) [38]. As with ABSTRIPS, HTNs enable greater efficiency in searching and more expressive solutions. Additionally, HTNs may be learned from data, but require considerable expert (and accurate) knowledge of the outcomes of subtasks to work [116]. How-

ever, these planning approaches assume deterministic transitions and are not applicable to MDPs and the assumptions they make.

Top-down approaches for MDPs include hierarchies of abstract machines (HAMs) [118, 119], hierarchical dynamic programming (HDP) [16], DetH* [17], and MAXQ-OP [14, 15]. HAMs define subtasks as finite state automatons (hence, machines) such that the policy one expresses is essentially a program of actions. HAMs adhere to typed abstract states (Action, Choice, Call, and Stop) providing the language necessary to achieve arbitrarily complex behavior; the drawback, however, is the immensity and complexity of the design burden for the machines. HDP leverages hierarchical maps (for robot navigation) as subtasks of an MDP, showing how a hierarchical variant of VI may be paired with state abstraction via clustering. DetH* similarly applies clustering to create abstract states, but instead pre-computes one high-level policy, which is then used to subdivide the ground MDP into subtask MDPs. MAXQ-OP computes an approximation of the MAXQ completion function for a given task hierarchy. Notably, it searches forward through state-action space, guided and limited by heuristics. In a way, MAXQ-OP turns the bottom-up problem of MAXQ into a top-down online search of plans.

Across these various methods, many limiting assumptions are made, such as a prerequisite that states may be clustered meaningfully (not split into too few or too many), or the requirement that a costly algorithm (in terms of computational complexity) such as VI must be used, or that the same algorithm is applied across all subtasks. The methodology I outline for hierarchies of abstract Markov decision process in the following sections circumvents these issues, to allow arbitrarily expressive state abstraction functions and subtasks defined independently, so that each subtask may use a separate planner most

appropriate to it.

7.2 Abstract Subtask Hierarchies

We introduce a new mechanism for hierarchical probabilistic planning: hierarchies of abstract Markov decision processes (AMDPs). The background and discussion presented here is based on the paper introducing AMDPs, which was work with Nakul Gopalan, Marie desJardins, Michael Littman, James MacGlashan, Shawn Squire, Stephanie Tellex, and Lawson Wong [54, 55]. In this collaboration, my main contributions were in the generation of results and their empirical assessment, as well as the initial design and definition of AMDPs.

The abstract Markov decision process encapsulates an MDP with state abstraction and termination conditions, defining a subtask relative to the ground MDP. Crucially, each AMDP possesses a local model (a reward function and transition probability distribution). This model operates over its abstract state-action space, without the need to access subtask models directly, the key difference between it and other methods. Thus, while AMDPs are most closely related to MAXQ, they go beyond its formalism by defining each subtask as a separate MDP. We contend that framing subtasks as MDPs in their own right offers the most natural way to regard them: when you have a decision to make, use a decision process. Thus, hierarchical decision making may be best represented with hierarchies of abstract subtasks, each its own MDP.

7.2.1 Abstract Markov Decision Processes

We define the abstract Markov decision process as a 7-tuple: $\tilde{M} = \langle \tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{T}, \tilde{R}, \gamma, \tilde{\mathcal{E}}, \phi \rangle$, relative to some ground MDP M. The AMDP $\tilde{M}$ has its own state space $\tilde{\mathcal{S}}$ consisting of states abstracted from M via the state abstraction function $\phi : \mathcal{S} \to \tilde{\mathcal{S}}$. The abstract actions $\tilde{\mathcal{A}}$ are either child subtasks (other AMDPs) or primitive actions in M. The abstract model is local, specific to the decision space of this subtask, $\tilde{T} : \tilde{\mathcal{S}} \times \tilde{\mathcal{A}} \times \tilde{\mathcal{S}} \to [0, 1]$ and $\tilde{R} : \tilde{\mathcal{S}} \times \tilde{\mathcal{A}} \times \tilde{\mathcal{S}} \to \mathbb{R}$. Additionally, we consider a set of terminal states, $\tilde{\mathcal{E}} \subset \tilde{\mathcal{S}}$, which contains all goal and failure states of the subtask. One assumption we make in this and subsequent discussions of AMDPs is that M is factored, so ϕ projects ground states into abstract state space typically by removing factors, aggregating states (mapping multiple s_m to the same $\tilde{s}$), or applying a nonlinear transformation (for instance, as specified by an expert).

The abstract Markov decision process (AMDP) framework provides a way to hierarchically decompose tasks into subtasks, each of which is represented as its own complete MDP with local state abstraction, reward, and transition functions [55]. Each node in the task hierarchy is treated as an encapsulated decision problem, differing from MAXQ's value function decomposition via completion functions, and the way in which an option pre-specifies its internal policy [150]. Successive child subtasks of AMDPs are designed to be smaller, more focused, and easier to solve than the ground MDP, as with other hierarchical approaches, by reducing the state and action spaces for any given decision. However, because AMDPs explicitly represent each subtask as a decision process, they permit all the relevant theory and practice for handling such problems. This difference is

the critical one among AMDPs and all prior approaches to subtasks: here, each subtask possesses all necessary pieces for planning and producing policies, with respect to only its immediate subtasks, independent of the ground MDP.

7.2.2 A Hierarchy of AMDPs

An abstract Markov decision process (AMDP) hierarchy $H = (N, E)$ is a directed acyclic graph of subtasks, derived from the MAXQ-style task hierarchy, decomposing a complex planning problem into a series of actionable subtasks. The internal nodes in N are AMDPs and the leaf nodes are primitive actions of the ground MDP. An edge in E indicates that the parent owns the child as a subtask; thus, the parent possesses a (symbolic) action in its abstract action space $\tilde{\mathcal{A}}$ corresponding to the linked child. One assumption is that AMDP hierarchies have one unique root AMDP, which serves as a subtask whose terminal states represent those of the ground MDP.

AMDP hierarchies produce optimal policies at each AMDP and, like MAXQ hierarchies, are recursively optimal given correct local state abstraction, reward, and transition functions. AMDP hierarchies enable top-down planning in stochastic environments, such that an agent plans only for subtasks that help achieve its main goal without computing plans for irrelevant subtasks. An AMDP is Markovian with respect to its own state-action space and transitions. The use of state abstractions, however, makes the abstract higher-level problems not necessarily Markovian relative to the base domain [13]. The execution of AMDP plans, thus, functions similarly to options in semi-Markov decision processes [150]. One consideration that arises from this property is the handling of failure

in subtasks. There is no inherent guarantee that non-goal termination conditions of child AMDPs are reflected in the state space of a parent AMDP. Thus, it is necessary when creating AMDPs that the designer ensures both the projection function and terminal sets are sufficiently expressive to capture such failure cases. Common forms of state abstraction should abide by five conditions that permit safety and theoretical properties outlined in Dietterich (2000) [34]. Similarly, a human expert must specify the components of each AMDP subtask (ϕ, failure and goal conditions for $\tilde{\mathcal{E}}$, and $\tilde{\mathcal{A}}$), but does not need to create the MAX and Q nodes as in MAXQ.

7.3 Planning with a Hierarchy of AMDPs

We now discuss planning with a hierarchy of AMDPs. In the context of the prior work discussed in this chapter, this approach may perhaps best be seen as top-down planning analogous to MAXQ. As with HTNs, we proceed from the root subtask; as opposed to MAXQ, we "complete" the value function within the confines of the AMDP itself. Because AMDPs include a local model and state space, their actions may be treated symbolically (i.e., executed in simulation according to the subtask model), a policy may be generated within that AMDP alone. To reiterate, given an AMDP, no decomposition to lower-level subtasks needs to occur. Thus, the first step in planning with a hierarchy of AMDPs is to create a plan at the most abstract level (the root node). For each subtask in the plan produced at the root, this process is applied again recursively for each next subtask. Once a subtask selects a primitive action as the next step, the agent actually executes this action and observes the true successor state. Combined, we attain an online

Algorithm 3 Planning with a Hierarchy of AMDPs

function START(Hierarchy H, MDP M, state s_0)
  SOLVETASK(H, M, ROOTINDEX(H), s_0)
function SOLVETASK(H, M, AMDP index i, s_t)
  $\tilde{M}_i \leftarrow$ GETNODE(H, i)
  if IS-PRIMITIVE($\tilde{M}_i$) **then**
    $a \leftarrow$ RESOLVE(M, $\tilde{M}_i$)
    $s_{t+1} \leftarrow$EXECUTE(M, a)
    $t \leftarrow t + 1$
  else
    $\tilde{s}_t \leftarrow \phi(s_t)$
    while $\tilde{s}_t \notin \tilde{\mathcal{E}}_i$ **do**
      $\pi_t \leftarrow$ PLAN($\tilde{M}_i, \tilde{s}_t$)
      $\tilde{a} \leftarrow \pi_t(\tilde{s}_t)$
      $j \leftarrow$ GETCHILDINDEX($H, i, \tilde{a}$)
      $s_{t+1} \leftarrow$ SOLVETASK(H, M, j, s_t)
      $\tilde{s}_{t+1} \leftarrow \phi(s_{t+1})$
      $t \leftarrow t + 1$
  return s_t

algorithm capable of tackling subtasks independently, all while under uncertainty.

Pseudocode is presented in Algorithm 3. The required input are a hierarchy, ground MDP, and some initial state. Planning then proceeds from the AMDP at the root index of the hierarchy. With each call to SOLVETASK, we check if the AMDP is a primitive subtask (the base case), and if so we find the corresponding primitive action from the ground MDP, and execute it. Otherwise, when the current AMDP is composite (non-primitive), we determine the current abstract state, create a plan, make a recursive call to that child subtask, and once control returns to this subtask, continue this process until a terminal state of this AMDP is reached.

I note that I have updated Algorithm 3 from the original we published in Gopalan et al. (2017) [55]. In addition to more detail, there are two significant changes. First, prior work relies upon a level-wise definition of AMDPs, such that each on the same level used

the same state abstraction function. The motivation for that requirement was to allow per-AMDP planners to be shared across AMDPs with the same abstract state space. I make no such restriction here, which is relevant for the new methods in the following chapter, such that I allow each AMDP to have their own ϕ; the trade-off is that an expert (or some programmatic mechanism) must be used to recognize and assign planners that can be reused among different AMDPs. Second, this pseudocode shows the decision to move the planning step inside the main **while** loop of SOLVETASK. This alteration allows for step-by-step replanning at the abstract level. Consider the case where the real execution (Line 7) returns a ground state that projects into the AMDP state space and results in a different abstract successor state than the policy expected. That is, when $\phi(s_{t+1}) \rightarrow \tilde{s}_{t+1} \neq \mathbb{E}_{\pi_t}[\tilde{S}_{t+1} \mid \tilde{S}_t = \tilde{s}_t, \tilde{A}_t = \tilde{a}]$. Thus, an updated policy may be computed based on the context of the newly observed state. To avoid unnecessary computation, π_t may be cached and reused if there is no difference in the next expected abstract state.

One possibility of the AMDP description of subtasks is their ability to represent parameterized or *lifted* subtasks. Using an OO-MDP or other factored state space makes this process straightforward. Let η be a list of state factors that may be parameterized, then H may include traditional AMDPs ($\tilde{M}$) and lifted ones ($\langle \tilde{M}, \eta \rangle$); in an OO-MDP these may be objects of a particular class. In the Taxi domain, we have lifted subtasks for NAVIGATE (parameterized over depot locations), and both GET and PUT (parameterized over passengers). Then, in the GETNODE subroutine of Algorithm 3, it generates all possible groundings of η and selects the relevant $\tilde{M}_i$.

7.3.1 Properties

I now elaborate upon some of the properties of planning with AMDPs, relating it in context to classic hierarchical RL and discussing optimality. The discussion I provide here expands on the more limited one in the original paper [55], and is based in part on my work for ELM (Section 6.4) [4] and PALM (Chapter 8).

Relation to Options

The most prominent method used for planning and learning in MDPs with a hierarchy of tasks is the options framework [150] (refer to Definition 6.1). As discussed, an option primarily defines a subtask, like an AMDP. Similarly, in practice, options are designed to move an agent through state space into or out of a "bottleneck" or landmark state. While AMDPs are closely related to options, they differ in that they explicitly maintain a local transition and pseudo-reward model, from which many different policies may be generated. AMDP subtasks are not locked into a policy, as with options, but are MDPs in their own right, permitting the use of any planning algorithm to produce a new policy at any time, independent of the other tasks. In making a decision, an option planner holds with the prescribed policy; AMDP planners *generate* their policy, and are correspondingly flexible and more robust to changes. In this sense, an AMDP is a formal tool for generating an option policy online. Thus, I contend that AMDPs are a natural way of framing subtasks of a hierarchy, embodying the guiding design principle that, whenever an agent has a decision to make, treat it as a full decision process unto itself.

Relation to MAXQ

The core difference between AMDPs and MAXQ is the latter's use of a derived model: transitions and rewards of subtasks are derived from the base MDP. While a MAXQ subtask is functionally analogous to an option, it exists under different structural and theoretical assumptions: a MAXQ agent leverages its hierarchy by learning the value function piece-wise for each node, expressed in terms of the smaller, more focused value functions of its children. MAXQ achieves a *recursively optimal* solution, such that it yields policies optimal at each level of the hierarchy, assuming its children are optimal [34].

AMDPs act as a kind of bridge between MAXQ and options, differing from both by treating each decision point in a hierarchical plan as a completely separate MDP, with its own state abstraction and local model of reward and transitions. In this sense, an AMDP serves as an SMDP relative to the ground MDP, with its actions functioning like options; to learn an AMDP model, thus, is to learn an option model. The consequence for planning with a hierarchy of local models is that, unlike planning with a hierarchy of derived-model MDPs like MAXQ or option models, every policy may be generated in an encapsulated way, expanding only those child subtasks that are absolutely necessary. In other words, an agent planning over a hierarchy of AMDPs never needs to solve subtasks that are not on the critical path to its goal, whereas both MAXQ and options must recursively compute even unrelated subtask solutions to determine how best to act. This property arises chiefly due to the top-down nature of the AMDP framework, and keeping this property motivates my next work on learning AMDP hierarchies and models.

Optimality

Dietterich (2000) covers types of optimality in the context of hierarchical RL, including recursive and hierarchical optimality [34], of which MAXQ and options are examples, respectively. The discussion is built upon previous work on the optimality of hierarchical policies [31, 120]. In brief, an algorithm that produces hierarchical policies may be regarded in separate degrees of optimality. One calls a policy *hierarchically optimal* when it yields the greatest return of any policy that could be specified by the hierarchy; assuming that the hierarchy's subtasks properly decompose the domain, this policy is typically also globally optimal with respect to the ground MDP. Dietterich defines recursively optimal policies, however, as those that are optimal only with respect to their given subtask. More precisely, they are optimal in the SMDP defined by the subtask relative to the ground MDP. The key difference is as follows: whereas a hierarchically optimally policy may take locally suboptimal actions to complete a subtask if doing so gives better results with respect to solving the whole task, recursively optimal policies act more greedily, completing each local subtask optimally to the potential detriment of the return of policy's rollout.

For AMDPs, I consider their optimality in this context, located in between that of options and MAXQ. Given a local model and state abstraction function, each AMDP subtask can be solved optimally, since it is simply an MDP unto itself. Hence any general AMDP hierarchy may produce recursively optimal policies. Yielding hierarchically optimal policies is not guaranteed in general. Consider a hierarchy for a multi-passenger Taxi domain: if the distance between depots is abstracted away from the root AMDP, then poli-

cies that GET and PUT passengers cannot be distinguished (GET-PUT-GET-PUT would look the same as GET-GET-PUT-PUT abstractly, though their grounded rollouts may be hugely different). Proving that a given AMDP hierarchy yields hierarchically optimal policies would require establishing certain invariants related to the influence of primitive actions on changing state factors in the ground MDP that are preserved in all AMDPs. In practice, however, it is often trivial to design AMDPs that yields hierarchically optimal policies (which is true for the expert-defined AMDPs discussed in the related work of Section 7.4).

7.4 Recent Examples & Results in Literature

I now provide an overview of AMDP planning in recent literature. In the original AMDP paper [55], we demonstrate its applicability in three highly stochastic domains: fickle Taxi, Cleanup World, and a situated robotics variant of Cleanup World (implemented on a Turtlebot). In the former two cases, our experimental results evince the value of AMDP hierarchies by decreasing the required planning budget from hundreds of thousands of Bellman backups to thousands. The experiments involving the robot demonstrate the wider applicability of AMDPs in real-world settings. For that domain, a continuous ground-level state space is incorporated into the lowest-level AMDP, offering a substantial challenge. Where options and MAXQ would require orders of magnitude more backups, the speedup afforded by the expert-defined AMDP hierarchy allows the Turtlebot to solve the task online, in real time.

Continuing in the context of robotic sequential decision making, AMDP hierarchies

can jumpstart learning dynamics for the manipulation of objects [78]. This work outlines and explores solutions for the tabletop organizational packing task, in which graspable objects may be picked, placed, and moved among containers on a table. They find that subtask transition functions are more easily learned from demonstrations on simpler 1-item 1-container tasks, by the inherent relation-based nature of the OO-MDP formalism, and that transfer to harder tasks is subsequently enhanced by the structure and reuse of AMDPs. Additionally, because of the high difficulty in exploring the transition space of such a robotic control domain, they contribute two novel representations of action selection (state-centric and action-centric). Combined and implemented in the execution of AMDP hierarchical policies, the resulting algorithm successfully finishes the more complex 4-item 2-container task as demonstrated on a physical robot.

Deep abstract Q-networks [131] extend the AMDP framework to the context of deep reinforcement learning, using neural networks as value function approximation. They demonstrate the applicability of an object-oriented AMDP planner and learner to handle sparse rewards, solving complex tasks with backtracking and better explainability than the standard deep Q-network algorithm.

Other research has explored the use of AMDP hierarchies in the setting of language grounding [10, 74, 75]. In particular, they consider the grounding task in which commands are expressed in natural language by a human observer and interpreted by a robot performing the desired actions. They apply AMDP hierarchies as a scaffold for subtasks in their proposed deep recurrent action-goal grounding network, or DRAGGN framework. Leveraging OO-MDPs, they describe how goals and actions may be represented semantically in terms of parameterized factored reward functions (that may bind to

various target objects in a domain). Given a textual command, a DRAGGN model interprets it in terms of a lifted reward function and degree of abstraction (level of an AMDP hierarchy), which are passed to a grounding module (in this case, a lookup of objects by identifiers), and then solved as an AMDP by incremental planning and execution. The intermediary lifted representation uses two separate neural networks; one for the "callable unit" and another for the binding arguments; one DRAGGN variant explores learning a shared embedding space for those two networks, while another DRAGGN keeps the learned embeddings independent. These DRAGGN approaches are compared with other language models (IBM2, a feedforward neural network, and both single- and multi-output recurrent neural networks), and excel in both the simulated Cleanup World and physical Turtlebot environments. Ultimately these efforts show the flexibility of AMDPs and their ability to generalize across related task, even in a complex scenario, for example, as an agent physically embodied in a robot directed to solve tasks by a person using natural language.

7.5 Conclusion

The previous chapter investigated bottom-up approaches to learning, while this chapter covers various top-down strategies for planning. At the core of my current discussion is a realization about the nature of hierarchical probabilistic planning: that we can apply recursion and consider each action we approach as its own proper subtask. In other words, in using AMDPs we are saying: treat every decision as its own decision process. This perspective is critical for my final synthesis of these ideas in the next chapter, investigating

how to combine abstract learning and planning in a comprehensive framework.

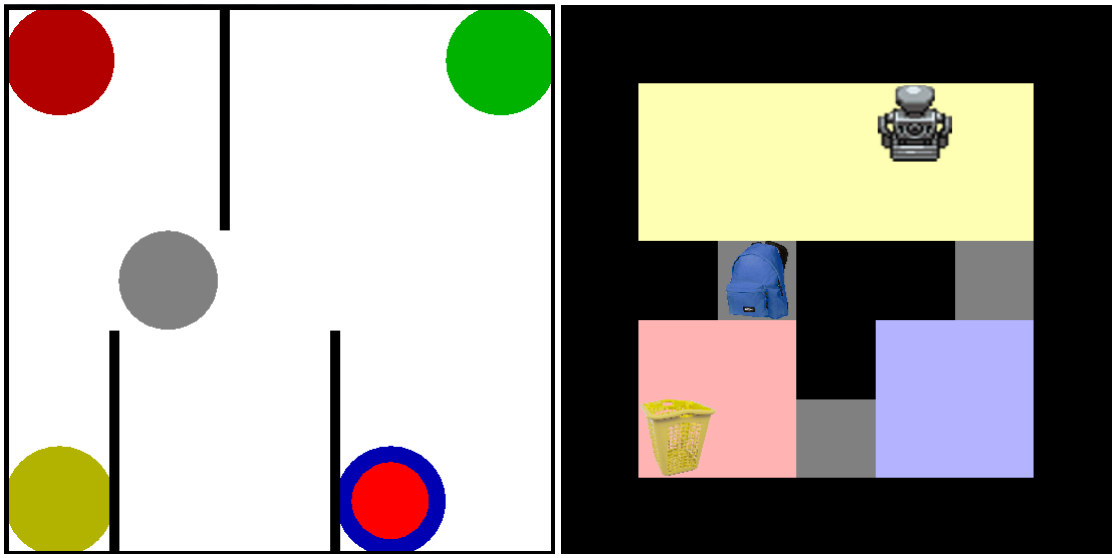
Chapter 8: Planning with Abstract, Learned Models

I introduce a novel approach for decision-making agents to first decompose a complex probabilistic planning problem into a hierarchy of tasks, and then learn models for all of the generated subtasks. The hierarchy can either be learned from experience or supplied by a human expert. Once equipped with a hierarchy, a model-based reinforcement learning strategy enables the agent to learn to produce plans that solve a large, stochastic, feature-rich environment where each subtask is represented independently as an abstract Markov decision process. My primary contribution consists of two general algorithms: one to construct hierarchies of tasks, and one to learn their models entirely from data. My approaches are flexible, able to integrate existing hierarchy- and model-learning algorithms together with any given MDP planners, as different domains may require. Moreover, they are scalable; combined, my methods learn to plan more efficiently than the most similar existing model-based hierarchical learning algorithm, R-MAXQ, achieving greater cumulative reward faster, while also offer a more generalized approach where learned subtask models are independent, able to be recombined or transferred.

8.1 Introduction

Complex human decision making requires reasoning abstractly over time and goals: recognizing, learning, and planning repeatable patterns of behavior that facilitate reaching a time-distant goal. Solutions to long-term tasks can be achieved by hierarchically composing temporally abstract subtasks that achieve intermediary subgoals that focus only on locally-relevant context and utilize actions at the appropriate level of abstraction. To handle temporal abstraction, previous efforts, such as the options framework [150] and MAXQ [34], have investigated extensions to Markov decision processes (MDP), the standard formalism for modeling probabilistic decision making. These efforts plan and learn using hierarchies of actions, tasks, and goals. They decompose complicated tasks into more manageable subtasks, then sequentially compose the subtasks to solve the initial task. A primary limitation of prior work is that either or both of the hierarchical structure and subtask models must be expert-defined, hand-crafted, or engineered in some way (the task hierarchy and policies, for MAXQ and options, respectively). Such a requirement is challenging, labor-intensive, and not guaranteed to produce a representation that best enables the agent to solve its goals or transfer knowledge to new, related problems.

To that end, I present two novel algorithms: one that learns to construct hierarchies of abstract Markov decision processes from data and another that learns the models for each subtask in the hierarchy. Together, these approaches reduce the burden on an expert to design tasks and engineer their connections and state abstractions. Furthermore, my results demonstrate that learned hierarchies can be more effective than expert hierarchies, outperforming the latter in certain instances. The core contribution of this work is the



(a) Example Taxi state. (b) Example Cleanup state.

Figure 8.1: Example OO-MDP states. In 8.1a, objects are the taxi (gray), walls, depots, and passenger (red). Objects in 8.1b include the agent, two blocks, three doors, and three rooms.

union of HA-Maker, which takes any general hierarchy-learning algorithm and converts its output tasks into abstract MDPs (AMDPs), and PALM, which applies model-based reinforcement learning to plan over the hierarchy of learned AMDPs. I demonstrate that my combined approach outperforms the most closely analogous model-based hierarchical learning algorithm, R-MAXQ [69].

The remaining content of this chapter is based upon Winder et al [171], and a publication currently in submission, leading from AMDPs in general to a more comprehensive model-based reinforcement learning approach, where every structure is learned from data. The discussion and experimental results I present include contributions from my fellow lab mates and students, Stephanie Milani, Matthew Landen, Erebus Oh, Shane Parr, Shawn Squire, and my advisor Marie desJardins.

8.2 Approach

I consider a goal-based interpretation of tasks where a single MDP defines one specific task, drawn from a distribution of MDPs (i.e., a family of related tasks). This task universe U is defined to encompass object classes, possible actions, physics governing how state variables may interact and change, and conditional rules characterizing goal states. An MDP may be sampled from this distribution, $M \sim U$. Given the actions and physics of U, specifying a starting state induces the specific T. The set of reachable states $\mathcal{S}$ follows directly by the transition rules of T. The R particular to this MDP may be derived depending on termination conditions (e.g., for goal and failure states), and rules or predicates may be defined to determine when reward should be affected (e.g., "apply negative reward on a transition from a state back to itself"). Importantly, learning a task's model aids the transfer of behavior to related tasks in the same U [3].

8.2.1 Related Work

I now reintroduce MAXQ as discussed in Section 7.1.2.1 from the perspective of learning models. The MAXQ hierarchical RL approach [34] leverages a *task hierarchy*, a graph from a root goal to child subtasks, decomposing the complex value function of the broad task into a set of smaller, more easily computable value functions, each with transitions and rewards derived from the base MDP. R-MAXQ [69] is a model-based RL algorithm that builds upon MAXQ. R-MAXQ learns partial models of each task in a given task hierarchy, recursively computing and storing models from subtasks to propagate knowledge of transition probabilities and rewards to the higher-level models. R-MAXQ applies

R-MAX [22] to approximate the rewards and transitions of primitive actions. To determine the model dynamics of composite tasks, modified Bellman equations relating the parent's model to the models of children are applied, decomposing recursively to the base actions approximated by R-MAX. After the knowledge of the ground MDP is propagated up the task hierarchy, the agent can plan at an abstract level using a greedy policy. After a bounded amount of exploration, the agent's hierarchical model will converge to a near-optimal solution. As with MAXQ, R-MAXQ achieves a recursively optimal policy, optimal at each level of the hierarchy assuming its children are optimal. By trading optimality for near-optimal solutions, recursively optimal methods offering significantly faster planning times; however, a poorly specified task hierarchy can result in suboptimal or unsolvable plans.

8.2.2 Model-Based Reinforcement Learning

In my novel method, Planning with Abstract Learned Models (PALM), an agent uses an AMDP hierarchy to plan a solution while learning the model of each task from experience. PALM simplifies the computationally intensive, "flat" planning problem in M by treating each node in the hierarchy as its own decision (represented as an AMDP), and then computing the AMDP task's policy locally. Conceptually, PALM plans the solution to a given task, selects a subtask, recurses, and continues this process successively until reaching and executing a primitive action (causing the agent to make a "real" transition in the ground MDP). It updates its task models based on the experiential data: the transitions observed as it executes actions in the ground MDP.

In PALM (Algorithm 4), planning and execution are interleaved and occur at multiple levels of abstraction. The inputs of PALM are the ground MDP M, an initial ground state $s_0 \in \mathcal{S}$, and a hierarchy H of AMDPs, which can be expert-defined or learned through an automated method like HA-Maker (Algorithm 5). Additionally, designers must select one or more planning algorithms (e.g. VI) and an algorithm for model-learning (e.g. R-MAX) to use in each AMDP.

In PALM, planning begins recursion on the root AMDP task. With each recursive call to PALM, s_t is projected into the given AMDP's state space by applying state abstraction, $\phi(s_t) \rightarrow \tilde{s}_t$. PALM then computes a local policy π for the AMDP and selects the next planned action, $\pi(\tilde{s}_t) \rightarrow a$. If that action is primitive ($a \in \mathcal{A}$), executing it in the base environment returns the next ground state, s_{t+1}; if it is a subtask, then it is an AMDP linked to the current one in the hierarchy. On completion of the execution step or recursive call, PALM abstracts the new ground state, $\tilde{s}_{t+1}$, obtains the pseudo-reward r for the abstract transition, and updates the current AMDP's model (recomputing the approximation of $\tilde{T}$ and $\tilde{R}$ based on the observed transition). In lines 18-21, the model-learning algorithm in each AMDP explicitly handles success and failure separately. For my implementation, in the case that $\tilde{s}_t$ is a goal or failure, r is overridden to apply the maximum or minimum value, respectively.

8.2.2.1 Nonstationarity

An abstract model learning on an unconverged lower-level model is following a non-stationary process. To avoid the effects of nonstationarity that would be introduced by

Algorithm 4 Planning with Abstract Learned Models

function START(Hierarchy H, MDP M, state s_0)
  Initialize models for all AMDPs $\tilde{M}_i \in H$
  PALM(H, M, ROOT-INDEX(H), s_0)
function PALM(H, M, AMDP index i, s_t)
  $\langle \tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{T}, \tilde{R}, \gamma, \tilde{\mathcal{E}}, \phi \rangle \leftarrow \tilde{M}_i \leftarrow$ NODE(H, i)
  $\tilde{s}_t \leftarrow \phi(s_t)$
  while $\tilde{s}_t \notin \tilde{\mathcal{E}}$ **do**
    $\pi \leftarrow$ PLAN($\tilde{M}_i, \tilde{s}_t$)
    $a \leftarrow \pi(\tilde{s}_t)$
    if IS-PRIMITIVE(a) **then**
      $s_{t+1} \leftarrow$EXECUTE(M, a)
    else
      $j \leftarrow$ CHILD(H, i, a)
      $s_{t+1} \leftarrow$ PALM(H, M, j, s_t)
    $\tilde{s}_{t+1} \leftarrow \phi(s_{t+1})$
    $r \leftarrow \tilde{R}(\tilde{s}_t, a, \tilde{s}_{t+1})$
    UPDATE-MODEL($\tilde{M}_i, \tilde{s}_t, a, \tilde{s}_{t+1}, r$)
    if TASK-COMPLETED($\tilde{M}_i, \tilde{s}_{t+1}$) **then**
      HANDLE-SUCCESS($\tilde{M}_i, \tilde{s}_t, a, \tilde{s}_{t+1}, r$)
    if TASK-FAILED($\tilde{M}_i, \tilde{s}_{t+1}$) **then**
      HANDLE-FAILURE($\tilde{M}_i, \tilde{s}_t, a, \tilde{s}_{t+1}, r$)
    $t \leftarrow t + 1$
  return s_t

making abstract updates before their subtask models have converged, I employ an update strategy based on the "knows-what-it-knows" (KWIK) framework. A KWIK algorithm (e.g. R-MAX [91, 151]) reasons explicitly about its transitions as either known or unknown. Although omitted from my pseudocode for succinctness, the recursive return step of PALM includes a signal indicating the known/unknown status of the transition that just occurred. This signal informs the parent task if it should ignore or process the transition in UPDATE-MODEL. In effect, subtask models are cemented before their abstract, parent models are learned. Experimentally without this update strategy, I found that PALM produces solutions swiftly while learning on nonstationary models, even exhibiting optimal

behavior; however, the aforementioned strategy ensures abstractive optimality and that PALM converges to a recursively optimal policy. All results reported in Section 8.4 follow the procedure of learning lower task models first before beginning to make updates to their parent AMDPs.

8.2.2.2 Complexity

I consider the complexity of each AMDP task individually. In the execution of PALM on a given AMDP, computational complexity is dominated by the planning algorithm. In the worst case, this planner is recomputed at each step. The sample complexity for an AMDP task in general will be $O(\rho)$, where ρ is the sample complexity of the model-learning algorithm used; for R-MAX, $\rho = |\mathcal{S}|^2|\mathcal{A}|V_{max}^3\epsilon^{-3}(1-\gamma)^{-3}$ given M's maximum value V_{max} and the PAC-MDP parameter ϵ [89]. Thus, hierarchies of fewer nodes and shallower depth are typically preferable in terms of sample complexity. Depending on the planning algorithm and model-learning algorithm, it is possible to cache planned policies between iterations, greatly ameliorating the computational cost of planning at each abstract step. In my experiments, I omit caching when comparing PALM with R-MAXQ and enable it for the PALM-only experiments.

8.2.2.3 Flexibility

PALM is agnostic to both the internal model-learning and planning algorithms (lines 8 and 17 of Algorithm 4, respectively). This property is one of PALM's crucial benefits: experts can specify different algorithms according to different decision problems solved by

Algorithm 5 Making a Hierarchy of AMDPs from Data

function HA-MAKER(Trajectories $\mathcal{D}$, algorithm H)
  $G \leftarrow \texttt{H}(\mathcal{D})$ ▷ H constructs a MAXQ-style task graph
  $H \leftarrow \emptyset$ ▷ Initialize an empty graph of AMDPs
  for subtask $g_i \in G$ **do** ▷ Reverse topological order
    $\langle \tilde{\mathcal{A}}, \tilde{T}, \tilde{R}, \gamma, \tau, \chi, \theta \rangle \leftarrow g_i$ ▷ Unpack the subtask
    $\phi \leftarrow$ EXTRACT(τ, χ, θ)
    $\tilde{\mathcal{S}} \leftarrow \emptyset$ ▷ $\tilde{\mathcal{S}}$ induced by ϕ, $\tilde{T}$ during execution
    Replace $g_j \in \tilde{\mathcal{A}}$ with the AMDP $\tilde{M}_j \in H$
    $\tilde{\mathcal{E}} \leftarrow \{s \mid s \in \mathcal{S}, \tau(s) \vee \chi(s)\}$
    $\tilde{M} \leftarrow \langle \tilde{\mathcal{S}}, \tilde{\mathcal{A}}, \tilde{T}, \tilde{R}, \gamma, \tilde{\mathcal{E}}, \phi \rangle$
    $H \leftarrow$ LINKNODE$(H, i, \tilde{M})$ ▷ Link $\tilde{M}$ to its children, the AMDPs in H matching the actions in $\tilde{\mathcal{A}}$
  return H

separate AMDP tasks. In my experiments, I have agents plan using either VI or bounded real-time dynamic programming [105], and subsequently follow a greedy policy, while handling the learning of models with R-MAX. I select VI and R-MAX to align my results with R-MAXQ (which uses them), and to demonstrate the utility of PALM with a stock approach under minimal assumptions. Other planners and model-based reinforcement learners could be used without changes to PALM's structure.

8.2.3 Hierarchy Learning

I introduce Making a Hierarchy of AMDPs from Data (HA-Maker), a direct procedure for constructing an AMDP hierarchy from solution demonstrations by leveraging existing hierarchy-building algorithms that yield task graphs. Algorithm 5 outlines HA-Maker's pseudocode. The first input is a set of trajectories, $\mathcal{D}$, demonstrating solutions to various tasks drawn from a domain distribution. I assume each trajectory is collected on $M \sim U$, an MDP sampled from the universe of possible tasks in the given domain. The second input is a hierarchy-learning algorithm, H, that yields a rooted, directed graph of tasks

$\texttt{H}(\mathcal{D}) \rightarrow G$, such that H learns the subtask dependencies, terminal conditions, and (optionally) the state abstraction. HA-Maker constructs AMDPs from tasks in the learned graph through a process that assembles each component in a reverse topological ordering, to build primitive AMDP tasks first, ending with the root AMDP. Notably, each subtask $g \in G$ consists of $\langle \tilde{\mathcal{A}}, \tilde{T}, \tilde{R}, \gamma, \tau, \chi, \theta \rangle$, which are, respectively, the task's own action set (either child subtasks or primitive actions), the transition probabilities for that task, the pseudo-reward function, the discount factor, the goal termination predicate, an optional failure termination predicate, and an optional set of factored state parameters based on OO-MDP classes. As in my experiments, $\tilde{T}$ and $\tilde{R}$ may be unknown or could be initialized based on prior knowledge; θ may be empty.

HA-Maker then initializes an empty graph of AMDPs, H. Starting from the child tasks and proceeding in reverse topological order, HA-Maker takes the following steps for each subtask in H. First, it extracts the state abstraction ϕ from the terminal predicates (τ, χ) and object parameters (θ) based on the attributes needed for objects present in them. Using HierGen (see Section 8.7) as H, θ contains only the objects causally relevant to the task, both τ and χ identify the factors needed to reason about goal and failure states, and thus the resulting ϕ abstracts away features not needed. For other H algorithms, if they do not afford a means to yield relevant state factors, then θ may simply contain all objects and thus EXTRACT will return all state factors (effectively, declining to use state abstraction). The set of abstract states, $\tilde{\mathcal{S}}$, for the subtask is initialized to $\emptyset$ because the states are not known beforehand. Reachable abstract states are induced implicitly by ϕ and $\tilde{T}$ during the model-learning of PALM on a given ground MDP. Second, any child subtask $g_j \in \tilde{\mathcal{A}}$ is replaced with the corresponding AMDP $\tilde{M}_j \in$ H, and terminal

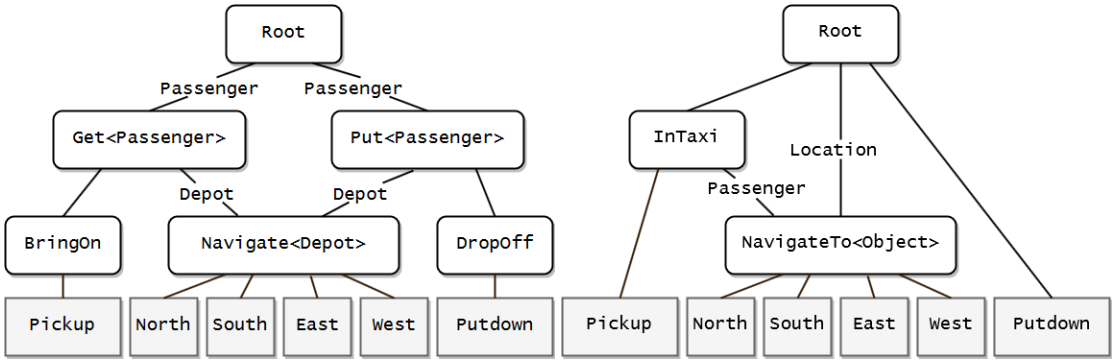
Taxi	Dimensions	Transitions	Fickle
Small	1x5	Deterministic	No
Classic	5x5	Stochastic	No
Classic Fickle	5x5	Stochastic	Yes
Large	20x20	Deterministic	No
Cleanup	**Dimensions**	**Blocks**	**Rooms**
Medium	7x5	1	3
Large	7x7	1	3
Small	5x5	2	2

Table 8.1: Domain variants used in the PALM experiments.

state membership is determined by τ and χ. Third, the AMDP $\tilde{M}_i$ is initialized with the standard tuple components. Finally, $\tilde{M}_i$ is added to the hierarchy by linking it to its children, the AMDPs in $\tilde{\mathcal{A}}_i$.

8.3 Experimental Methodology

I compare the performance of a PALM model-learning agent to the most analogous existing model-based hierarchical method, R-MAXQ. I also demonstrate the value of HA-Maker by comparing the performance of PALM with HA-Maker-generated hierarchies against PALM with expert-defined hierarchies. For HA-Maker, I select HierGen because it follows a principled causal approach to generate both task linkages and their concise state abstractions, where only the variables that are causally relevant are preserved [107]. Each PALM experiment consists of 20 independent trials of 300 episodes per trial (or otherwise specified), each with 10^5 maximum steps, and reward functions scaled to have 1.0 as the maximum goal reward. VI and R-MAX serve as the planner for PLAN and model-based reinforcement learner for UPDATE-MODEL (lines 8 and 17 of Algorithm 4).



(a) PALM-ET, the classic expert Taxi hierarchy. (b) PALM-HT, generated by HA-Maker, learned via HierGen [107]

Figure 8.2: AMDP hierarchies for the Taxi domain. Edge labels specify the parameters passed from parent to child.

Additionally, for some experiments I include a baseline by running Q-learning (QL), a flat reinforcement learning algorithm [167], to establish the relative value of having a hierarchy and indicate the difficulty of learning from scratch. As a model-free algorithm, QL does not maintain approximations of T or R and is thus unable to plan over its actions.

8.3.1 Domains

The Taxi domain [34] is a discrete environment with a taxi agent, passengers, depot locations, and walls positioned on a grid-world. The agent must deliver each passenger to its goal depot. The action space contains four navigational actions, north, south, east, west, as well as two passenger-parameterized ones: "pickup" for collecting a co-located passenger, and "dropoff" for depositing a held passenger at a depot location. Movement can be stochastic, correctly transitioning with a probability of 0.80 (else moving in a perpendicular direction). The *fickle* variant permits any passenger to change their desired destination stochastically on a movement action with probability 0.05.

I use the Cleanup domain as described in Chapter 5, but with different configu-

rations of rooms and blocks. I emphasize that Cleanup tasks appear deceptively simple: they present a combinatorial explosion of state space as more blocks and rooms are added, and the transition dynamics present many bottleneck "corner cases" in state space (when blocks are stuck in corners) that require a precise sequence of actions to exit. For example, a 7x7 three-room two-block task has approximately 18000 states; with four blocks, the MDP has several million states. The Cleanup domain makes use of a sparse, flat 0/1 reward structure (zero until a goal state is reached). I standardize this goal-based variant of Cleanup as follows: all blocks shaped like bags or backpacks must be placed in a room matching their color, while other blocks are irrelevant and serve only as obstacles.

8.3.2 Hierarchies

I describe the hierarchies used in this paper:

PALM-ET (Figure 8.2a) is based on the classic MAXQ task hierarchy [34]. The GET and PUT tasks of PALM-ET are parameterized over the passengers, abstracting away the Cartesian coordinates of the taxi, passengers, and depot locations.

PALM-HT (Figure 8.2b) is the AMDP hierarchy constructed by HA-Maker using HierGen to learn a hierarchy for Taxi from 30 solution trajectories. Its structure matches the one reported in a previous work [107]. Notably, this hierarchy possesses a ROOT task with the primitive "putdown" action, an INTAXI task (complete when the passenger is in the taxi), and a NAVIGATETO task (complete when the taxi is at the parameterized passenger's goal).

PALM-EC (Figure 8.3a), the expert hierarchy for Cleanup, has its root AMDP plan

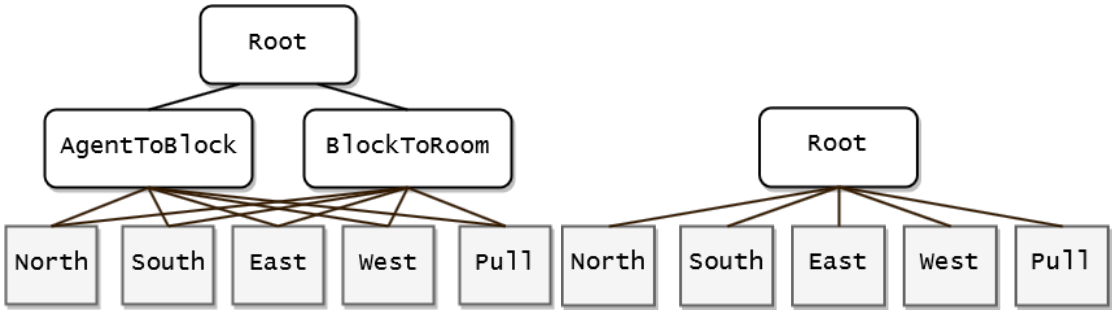
in terms of moving the agent to the block (A2B) and moving a block to a room (B2R). As an action in ROOT, B2R can only be initiated when the agent is adjacent to a block; similarly, A2B terminates when next to a block. A ROOT policy would thus generally require A2B then B2R for each block that needs to be moved

PALM-HC (Figure 8.3b), the learned Cleanup hierarchy, degenerates into a flat hierarchy without subtasks beyond ROOT, which contains only the primitive actions. I find this negative result surprising, arising from my use of the hierarchy learner HierGen as `H` in HA-Maker, and discuss it further in Section 8.5.

PALM-AC (Figure 8.3c), an amended, second-draft expert hierarchy for Cleanup is informed by PALM-HC. Each primitive action is wrapped in an AMDP task parameterized by the x-y coordinates of the destination relative to the agent position. Additionally, I define four LOOK tasks (one for each movement primitive) that permit the agent to face a block before pulling it. Combining these parameterized, shielded primitive actions with state abstraction over irrelevant variables means that a PALM agent will never plan an action that is illegal or results in a self-transition.

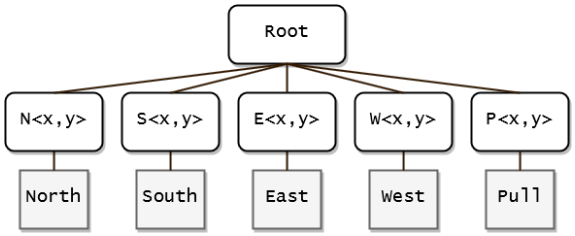
8.4 Results

I examine the utility of HA-Maker in comparison with expert-specified hierarchies, and I consider the performance of PALM in terms of the cumulative steps taken and reward acquired across episodes. The figures are shown with the 95% confidence region shaded.



(a) PALM-EC, the first expert Cleanup hierarchy.

(b) PALM-HC, generated by HA-Maker using HierGen.

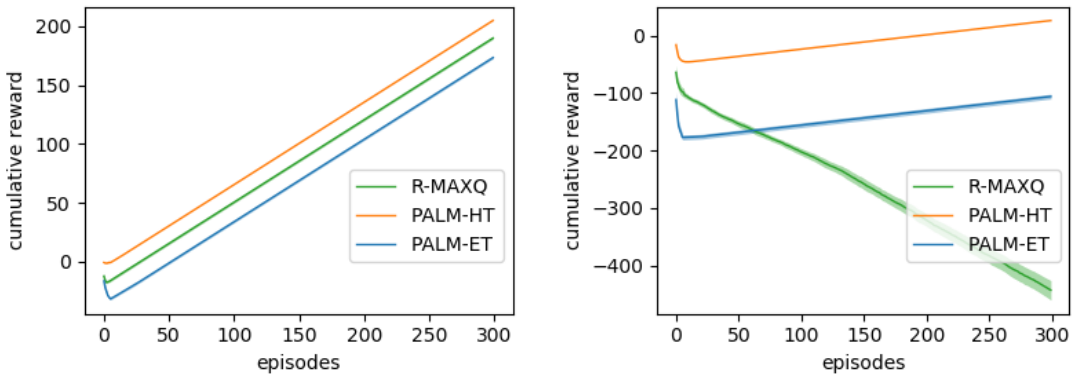


(c) PALM-AC, an amended expert hierarchy, informed by PALM-HC.

Figure 8.3: Hierarchies created for the Cleanup domain.

8.4.1 PALM & R-MAXQ

Figure 8.4 compares the PALM methods with R-MAXQ. In the Small Taxi domain, with deterministic transitions, R-MAXQ exceeds PALM when using the same hierarchical structure (PALM-ET). Consider results on the classic Taxi problem in Figure 8.4b. One may observe a different asymptotic relationship among R-MAXQ and the PALM methods under these conditions, such that R-MAXQ struggles to compute the recursive models of its subtasks even on this domain, with only 100 states. While I anticipate R-MAXQ ultimately approximates correct models to plan more efficiently in episodes beyond those shown, it will not surmount the results of PALM-ET and PALM-HT, both of which rapidly converge to the optimal policy.



(a) Small, deterministic Taxi.

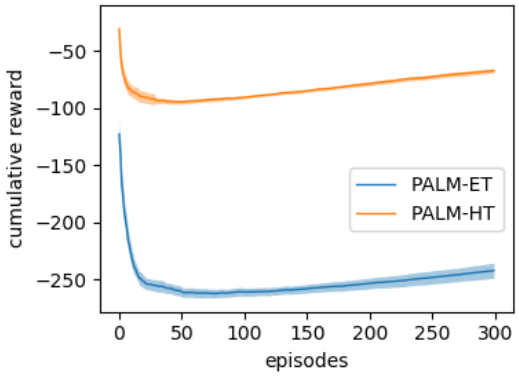
(b) Classic Taxi.

Figure 8.4: R-MAXQ, PALM-ET, and PALM-HT on Small Taxi and Classic Taxi. PALM-HT achieves greater cumulative reward for both domains.

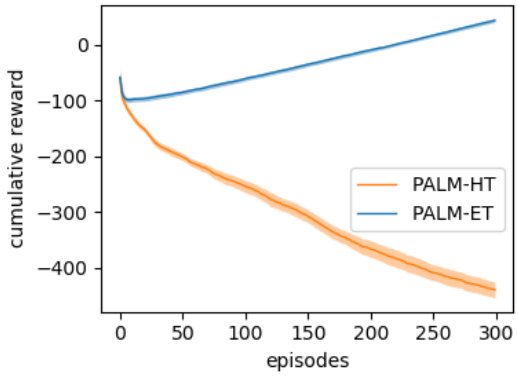
8.4.2 PALM with Expert & PALM with HA-Maker

It is observed that, for the fairly simple domains in Figure 8.4, PALM most directly learns the models for the hierarchy built by HA-Maker (PALM-HT). Regarding Figure 8.5, PALM-HT also succeeds significantly over PALM-ET in the version of the Classic Fickle domain with one passenger. However, PALM-ET achieves greater cumulative reward in the successive experiments than PALM-HT, with the latter faltering to increasingly greater extents as more passengers are added. I elaborate on this trend in Section 8.5. Figure 8.5d exhibits the case of a transferred model, where a converged model for the NAVIGATE AMDP is given to a PALM-ET agent and compared against PALM-HT.

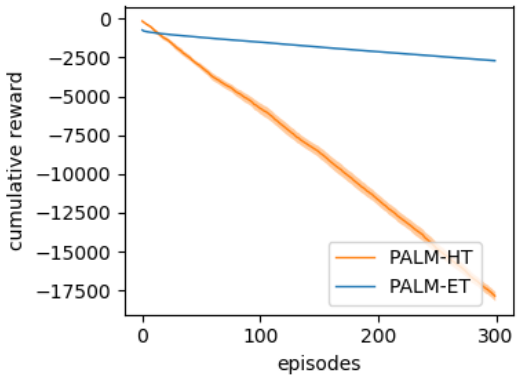
For the Cleanup domain, with results in Figure 8.6, it can be observed that the relative changes in cumulative steps rather than reward (since each complete episode yields a reward of exactly 1.0) as complexity is varied among tasks. Specifically, I consider when there are more rooms, Figures 8.6a and 8.6b, or fewer rooms but more blocks, Figures 8.6c and 8.6d. The asymptotic relations among the PALM methods hold across all



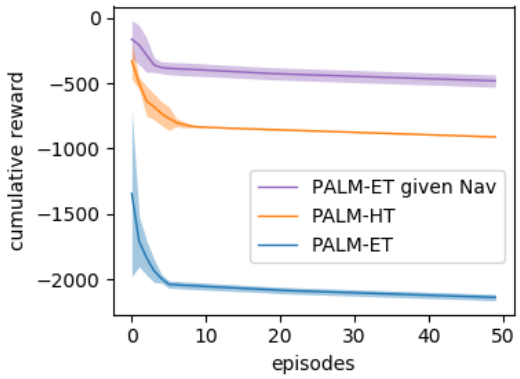
(a) Classic, 1 fickle passenger.



(b) Classic, 2 fickle passengers.



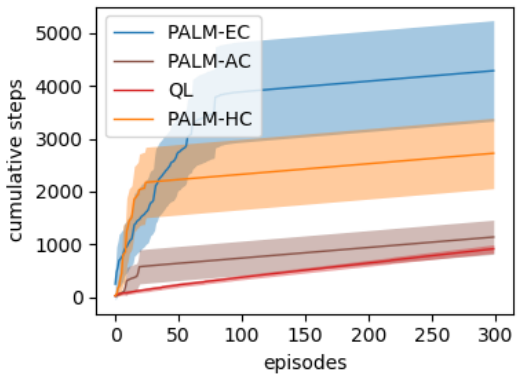
(c) Classic, 4 fickle passengers.



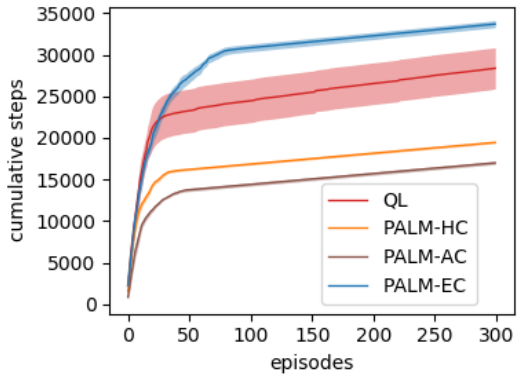
(d) Large, 1 passenger.

Figure 8.5: Cumulative reward for the Taxi domain. In 8.5d, I provide a PALM-ET agent with a converged model for NAVIGATE to show the benefit of transferring learned models.

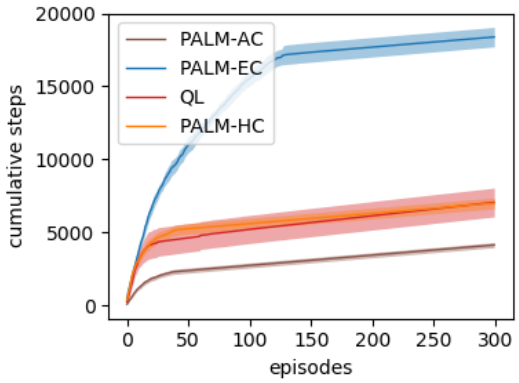
variant tasks: PALM-EC requires the most cumulative steps, while PALM-HC (PALM with the hierarchy constructed by HA-Maker) is in the middle, and the redesigned expert PALM-AC solves domains most readily. Both PALM-HC and PALM-AC outperform QL, which is learning directly rather than planning over learned models. In comparison, the initial expert hierarchy seems less effective, taking statistically significantly more steps to learn than QL, despite having fewer models than PALM-AC.



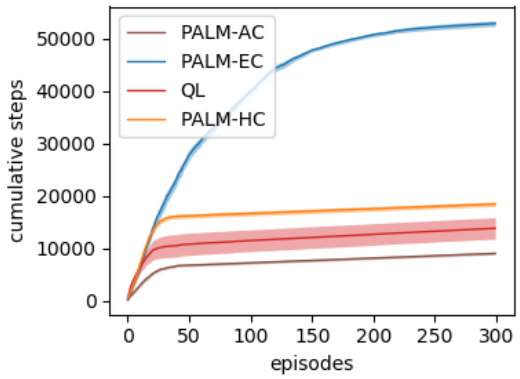
(a) Medium, 1 block, 3 rooms.



(b) Large, 1 block, 3 rooms.



(c) Small, 2 blocks (one irrelevant, one target), 2 rooms.



(d) Small, 2 blocks (both targets), 2 rooms.

Figure 8.6: Cumulative steps for the Cleanup domain. Note: fewer steps are preferable (achieving the goal reward faster).

8.5 Discussion

In brief, the value of PALM over R-MAXQ is its facility for scaling to increasingly complex domains while maintaining the ability to transfer its models. For HA-Maker, I find it can construct hierarchies that, in combination with PALM, outperform expert hierarchies. These learned hierarchies of AMDPs, however, come with the caveat that their structure may lead PALM to encounter difficulties and struggle in learning abstract models as task complexity increases.

8.5.1 PALM & R-MAXQ

The advantages of PALM relative to R-MAXQ are made evident by the results on the Taxi domain with respect to number of cumulative reward and scalability: as stochasticity and more states are introduced, R-MAXQ requires excessive computation in comparison to PALM. Assuming the same planning scheme is deployed in PALM and R-MAXQ (I use in both VI), PALM requires fewer Bellman updates for each task node in the hierarchy. This advantage arises because PALM's PLAN step (line 8 in Algorithm 4) needs only consider local transitions across the (abstract) state space of the AMDP for its task, whereas the analogous COMPUTE-POLICY step in R-MAXQ (Algorithm 2, [69]), must create a planning envelope of possible future states after each execution, recomputing the models of all children down to the primitive actions. Since R-MAXQ must calculate transition probabilities and rewards recursively down through all subtask models each time it plans for a given task, an increase in state space amplifies the effects of stochasticity across levels of the hierarchy. For PALM, since each model is computed independently, the effects of stochasticity are limited. In effect, because the AMDP framework treats each subtask as its own local decision problem, PALM can learn the value functions of these tasks in a more focused manner.

Exploration of R-MAXQ and PALM is also fundamentally different. While R-MAXQ is learning its models, the exploration guided by R-MAX is affected by the need to compute down to the primitive level; for PALM, exploration is handled irrespective of subtasks, because each AMDP that is being modeled has its own R-MAX. Though reasonable in such small domains, my cursory investigation of R-MAXQ on much larger, more

complex MDPs indicates that the intertwined nature of R-MAXQ's computation serves to exacerbate the scalability issue, whereas PALM's encapsulation of models inhibits such problems. I also note that my implementation of R-MAXQ followed the original specification [69], and while alternative design choices may help alleviate scaling issues, the structure of PALM's pseudocode roughly parallels that of R-MAXQ.

8.5.2 Independent Models

Perhaps the biggest difference between PALM and R-MAXQ, as well as MAXQ and options more generally, is that PALM yields completely independent models. Specifically, each subtask is represented as an AMDP that, when planning, is functionally just an MDP. This notion of treating abstract tasks as their own decision problems is one of the core insights from prior work on planning with AMDPs [55]. That is, in learning an abstract model for an AMDP, PALM is not creating interdependent "puzzle piece" forms of temporal abstraction that must link together precisely. Rather, PALM performs model-based reinforcement learning directly, just on abstract problems, where the result is not a static policy or fragment of a value function, but an encapsulated MDP in its own right. Thus with AMDPs, PALM can learn one MDP and transfer sufficiently abstract models to new, related tasks pulled from the same universe, greatly accelerating overall performance.

I include Figure 8.5d as an example of transfer, and refer to standard metrics for transfer in reinforcement learning [154]. To transfer in a model, it is first necessary to obtain either an expert-defined one or a learned one acquired via training on an MDP sampled from the same universe. In this case, tasks are drawn from the Large Taxi do-

main, for which the location of depots remains constant but the position and destination of passengers varies. Transfer is valid because the state abstraction of NAVIGATE is sufficient: it abstracts away passengers, so it is directly valid in any task once learned. This kind of encapsulated transfer is impossible with R-MAXQ because, even after it learns an abstract model, those transition probabilities and reward function depend ultimately on the R-MAX approximation on ground states specific to the given MDP. Provided with the converged model, the "PALM-ET given Nav" agent achieves an immediate and statistically significant jumpstart over the algorithm with the same hierarchy, PALM-ET, as well as having a shorter time to convergence and greater total reward. One can see this performance because NAVIGATE dominates the sample complexity, so possessing a correct model alleviates the burden to explore at that level, allowing the agent to more rapidly advance to learning GET and PUT. While introducing PALM here, I anticipate exploring the benefits and potential cases of transferring learned abstract models more extensively in future work, such as with value function approximation as discussed in Section 8.6.

8.5.3 Expert & HA-Maker Hierarchies

My main finding in pairing HA-Maker and PALM is that not only is it possible to learn every piece required for planning with hierarchies of abstract Markov decision problems entirely from data, but such hierarchies can rival the efficiency of those engineered by human designers. Figures 8.4b and 8.5a evince this result most clearly, where PALM-HT attains roughly the same asymptotic performance as PALM-ET in only a fraction of the steps required (that is, with less negative reward). The solution trajectories HA-Maker

requires can be gathered from random walks in instances of the ground domain, and in leveraging any task graph learner, HA-Maker can build and configure all the AMDPs needed for hierarchical planning, including state abstraction and a factored pseudo-reward function. These results likewise indicate PALM can take learned hierarchies and independently learn all of the remaining components of the AMDPs. Effectively, HA-Maker and PALM make it possible to create everything needed for hierarchical planning via sampled experiences without expert knowledge.

I uncover a problem of using learned hierarchies that is best illustrated by Figures 8.5b and 8.5c. In these tasks, the number of passengers is increased beyond the original number of passengers (one) in the trajectories for which PALM-HT was created. Where PALM-ET is robust to the added challenge, PALM-HT exhibits severely degraded performance. As with R-MAXQ in Figure 8.4b, I know that PALM-HT will eventually learn its models and be able to produce an optimal policy. But, the presence of fickle passengers causes PALM-HT to vacillate more heavily, due to its task parameterization. It must reason about (and explore) the combinations of moving among all passengers and locations in its ROOT task. Lacking a model for a PUT-style subtask, solving the PALM-HC ROOT requires handling all the ways held passengers could be "Putdown" at locations. Thus the utility of PALM and HA-Maker may be greatly lessened when attempting to transfer the hierarchy to increasing degrees of complexity in the universe of tasks to which they are applied.

For Cleanup, where I supply trajectories sourced from a variety of Cleanup variants (Table 8.1), the hierarchy learned by HierGen collapses all primitives into one task. Specifically, with any amount of variety in the state space of tasks comprising the so-

lution trajectories (with all tasks in a similar universe), HierGen is not able to parse its causally-annotated subsequences into a precedence graph of more than one node. In essence, this finding emphasizes the immense challenge of generalization that still endures for hierarchy-learning algorithms. Thus, for PALM-HC, HA-Maker only assembled one AMDP. Its representation is roughly equivalent to a flat model-based learner (though it is equipped with learned state abstraction, goal, and failure conditions). Despite degenerating to a single-task hierarchy, PALM-HC's performance in the Cleanup experiments does demonstrate that the union of hierarchy-learning and model-learning algorithms can produce valid, efficient solutions. All PALM methods in Figure 8.6 arrive at the optimal policy, which QL does not (yet) find in 300 episodes. In each case PALM-HC converges faster than PALM-EC, in many fewer steps and episodes. The effectiveness of PALM-HC over PALM-EC indicates it can leverage the mechanisms of AMDPs to reach the same optimal behavior through more efficient sampling. In contrast, while PALM-EC takes significantly more steps to converge, it is also produces a potentially more useful set of models, transferable to related tasks where those of less complex hierarchies are not.

The final hierarchy, PALM-AC, is created based on my observations of the first two; the fact this second iteration expert hierarchy performs better after observing a learned one suggests the value of cooperating with learned methods. Thus, an insight from this work is that enhancing expertise with knowledge gleaned from autonomously generated structures can lead to an approach that surpasses either of those on their own. I hope in future work to develop more sophisticated hierarchy-learning methods that can automate these insights.

8.6 Future Work

In the process of creating a comprehensive framework capable of dually learning hierarchies and their models, I encounter several challenges, sometimes unexpected and counter-intuitive. Above all, it is challenging to create a valid hierarchy - either by hand or learned from data. For instance, most hierarchical methods consider segregating actions; HierGen relies on the ability to parse actions based on their causal effects. Cleanup is a particularly challenging domain for this approach because "problem" states continually arise that require all five primitive actions. For example, when a block is in a corner, the agent must enter a state facing towards the block, pull to get it away from the wall, then navigate behind it. It is difficult to design subtasks that do not encounter such corner cases at some point. Candidate hierarchies may also inadvertently specify impossible subtasks when grounded to a new domain. Consider the B2R task of PALM-EC, with the goal of moving a block to a room. Grounded to a cross-shaped Cleanup state, where there is a single-cell room in the center, it is not possible to perform B2R to move a block from one junction of the cross to a perpendicular arm. Thus, future work must address the problems of recognizing when to replan, not being fully committed to a task (by terminating probabilistically), and both growing and pruning candidate tasks while learning models.

An advantage of the modularity afforded by my HA-Maker and PALM approach is the ability to use representational tools such as value function approximation for navigational or perceptual tasks in a low-level continuous state space, while using tabular, dynamic programming in higher-level AMDPs. Moreover, an abstract hierarchy could

be trained on a wholly symbolic hierarchy, with a discrete navigation AMDP, and then transferred over to a continuous domain, where only navigation needs to be relearned.

8.7 Related Work

Prior efforts in learning hierarchical structures of tasks include several approaches: discovering transferable option policies directly [23, 159]; learning high-level skills and abstract actions from demonstrations [82, 83]; and, causally annotated approaches that produce MAXQ-style task graphs. For the discovery of transferable options, policies are often broken down into sub-components that are reusable in novel domains and environments. In contrast, skill discovery centers more on the representational question, with the aim to learn high-level symbols that induce build minimal, abstracted MDPs representing options that chain from one to another. In the chaining of skills, the complexity of learning the global task's value function is circumvented by needing only to learn the smaller value functions of chained skills. The latter thread of research stemming from MAXQ includes VISA [70, 165], in which the causal effects of actions on state factors are modeled as dynamic Bayesian networks. In particular, each action receives its own network that captures the probabilistic change from a source state to successor state (on one time step). In the network, each state factor is a random variable, with edges indicating a temporal dependence among factors from one state to the next, capturing the causal connection of factors and the action to represent how they evolve probabilistically over time. HI-MAT [108] expands on this idea further by constructing a hierarchy of tasks based on the casual annotation of a single successful trajectory through state space. Its

extension, HierGen [107], outlines a more general framework that constructs a hierarchy from many trajectories. This latter group of techniques most closely aligns with the AMDP framework, as their task hierarchies can be converted directly to hierarchies of AMDPs, given an explicit set of terminal goal and failure conditions.

8.8 Conclusion

PALM demonstrates the viability of a model-based approach on the AMDP framework for hierarchical planning, where entirely abstract models are learned from experience gained and propagated across the abstract level itself. With HA-Maker, I show how to deploy existing structure-learning methods to learn task hierarchies that can be converted to an AMDP hierarchy, yielding a compact and parameterized representation. With PALM, an agent can approximate and plan over the models that best suit each task in a hierarchy of decision problems. PALM maintains the benefits of planning with AMDPs, such as increased scalability and transferability of models. United, they grant a unique capacity to learn, from sampled experiences, every component of tasks and their models needed for hierarchical model-based reinforcement learning, all without requiring any human engineering of reward structure or expert-defined transition dynamics. Thus, planning with abstract, learned models provides a complete framework for learning, bottom-up from data, the structures necessary to perform top-down, hierarchical probabilistic planning.

Chapter 9: Conclusion

My aim has been to create more intelligent agents, capable of generalizing their experience beyond the narrow set of conditions upon which they were trained. With anomaly reasoning through concept formation, I outlined how uncertain and novel objects in an agent's environment may be interpreted. By leveraging the theory of formal concept analysis, I demonstrated how decision-making algorithms can be made concept-aware to better adapt to anomalies, generalizing by transferring behavior for previously learned concepts in new circumstances. To extend agents' reasoning over longer time horizons, I expanded the field of hierarchical reinforcement learning with new methods for bottom-up learning of abstract subtasks. Taking the converse paradigm, I discussed structures for efficient top-down hierarchical planning. Synthesizing these forms of abstraction together, I developed and analyzed a novel framework for integrated planning and execution, where agents themselves learn the tools they need to create state and temporal abstractions that enable transfer and generalization.

9.1 Summary of Contributions

This dissertation covered an array of contributions in achieving more general, adaptable abstract decision making.

I began by discussing the notions of *concepts* and *habits* as abstractions of states and actions, respectively. I then outlined the problem of anomaly reasoning, motivating the use of formal concept analysis to generate an explicit, hierarchical conceptual structure of knowledge. From this, I contributed a **theoretical framework for anomaly reasoning**, decomposed into the processes of identification, interpretation, and adaptation. I formulated interpretation as a classification problem, including an interpretation prototype featuring a **lattice abstraction procedure**. In the course of an empirical analysis, I also produced a novel domain, the **NetHack Monster data set**. This data is extracted and cleaned from NetHack's game files, in a format suitable for supervised learning, to be made available for other researchers.

In terms of concepts, I developed **concept-aware feature extraction (CAFE)** to create formal concept-based usable in function approximation and state abstraction. An earlier version of CAFE is published in Winder and desJardins (2018) [170]. I articulated the framing of CAFE in both a **contextual bandits with concepts (CBC)** and a **concept-aware reinforcement learning (CARL)** setting. The analyses of CBC and CARL, with the updated definition of CAFE, are in preparation for submission as conference paper. I also defined the **concept meta-graph** visualization; the accompanying software to render it will be available alongside the CBC domains upon their release.

From the perspective of habits as subtasks, I surveyed a history of bottom-up learning and top-down planning methods. In Chapters 6 and 7, I discussed my collaborations to make more adaptable algorithms for learning and planning with temporal abstraction. In addition to substantial writing, my contributions to these projects include the creation and empirical assessment of a more sample-efficient approach to option learning, the

expected-length model (ELM), and the analysis of our novel technique for transferring abstract policies, **portable option discovery (POD)**. A paper on the former will be published this year [4], while the discussion in this thesis of the latter is based upon Topin et al. [159]. I then provided an overview of the context for top-down planning with **abstract Markov decision processes (AMDPs)**, where each separate subtask is a decision problem unto itself. For this work, I primarily contributed to the algorithmic design and the generation of experimental results; our main work is published in Gopalan et al. [55]. Bringing together these top-down and bottom-up paradigms, in Chapter 9, I articulated a new model-based reinforcement learning algorithm, capable of learning hierarchical structure and then individual AMDP subtask models: **planning with abstract, learned models (PALM)**. This work is based on an simpler approach published in a workshop paper [171]; I am preparing a more extensive paper for submission to a conference later this year.

For my implementations of algorithms and experiments, I built upon the Brown-UMBC Reinforcement Learning and Planning (BURLAP 3.0) library [97].[1] The substantial novel additions and changes to the original BURLAP source code warranted a forking of the project into five separate packages:

- **HessianJ**: the main fork of BURLAP, containing POD, a suite of new domains, and revised source code fixing several major bugs.
- **FleeceJ**: general contextual bandits domains and the LinUCB variants.
- **CanvasJ**: algorithms and data structures for Formal Concept Analysis, CAFE,

[1] https://github.com/jmacglashan/burlap

CBC, CARL, and the concept meta-graph visualizer.

- **LampasJ**: abstract Markov decision process data structures, ELM, and PALM.
- **TulleJ**: standardized simulation code for running experiments loaded from configuration files.

These repositories collectively form the **FabricRL** project, a branch of and successor to BURLAP.

Due to the size of FabricRL and its importance to the results I present in this thesis, I must reiterate my thanks in particular to James MacGlashan, the main author of BURLAP, for his great efforts on that library, which helped to make mine possible. Likewise, I wish to thank Nakul Gopalan and Matthew Landen, both of whom authored significant pieces of the AMDP and PALM architectures, and related code, upon which the versions included in FabricRL are based.

9.2 Future Work

I have investigated two parallel threads of research in this dissertation, state abstraction and temporal abstraction. In demonstrating how both enable transfer and generalization of artificially intelligent agents, I have laid out a plan for future work to build upon, and ultimately, synthesize them more fully. One way to achieve this goal would be to unite a CAFE-like concept-forming, anomaly-reasoning framework with a PALM-like bottom-up learning of top-down structures for planning. With the remaining space, I sketch an outline of possible future directions to explore.

Larger, More Complex Domains

NetHack is a single-player video game in which the player must collect items and defeat enemies to escape from a multi-level dungeon. The immense variety of object types and possible actions makes transferring knowledge and adapting to novelty necessary.

An agent playing NetHack often encounters new objects functionally similar to learned classes of objects (but differing by some specific attributes). In combining CAFE with a model-based hierarchical RL algorithm like PALM, transferring concepts would facilitate an agent's ability to respond to unseen types of items, furniture, and enemies, such that subtasks could be parameterized by these abstract features.

In applying concept formation to domains like NetHack with essentially open-ended state spaces, a central challenge that consider is the ever-increasing space concepts. Thus, I expect heuristic methods for pruning the concepts to consider, would be especially useful. One might imagine learning a scoring function based on minimum description length (MDL) that finds the optimal compression of a given (A)MDP state space, characterized by a concept meta-graph of only the most relevant concepts.

In partially observable environments like NetHack, agents have limited access to the underlying state, leading to uncertainty in decision making. Often, they have only a set of observations and must use these to update their beliefs about the state they currently occupy. Since CAFE maintains a general notion of its input, concepts can just as easily be derived from belief states. (CAFE makes does not require access to a state's full set of features.) Thus, CAFE's facility for anomaly reasoning makes it directly applicable to the central challenge of POMDPs. Learning transfer in POMDPs is the same as in the

fully observable case, preserving known concepts and their weight parameters θ from a source POMDP to a target POMDP. In transferring from domains with more information to ones with less (such as training on a fully observable MDP and deploying on a related POMDP), concepts could allow inference of the latent, unobserved features.

Concepts could form the basis of memory that is used to persist context across time in POMDPs. Concepts will be stored in a read-write format, similar to neural maps but more directly analogous to sparse distributed memories, where their binary representations serve as addresses into memory space. Given a POMDP state, an agent's observation maps to a behavior by extracting concepts via CAFE, indexing those into memory, and interpreting the stored knowledge of good and bad actions based on the co-occurrence of those concepts. Crucially, where the existing memory techniques rely solely on the literal, grounded features of the state, my proposed concept-based memory differs by leveraging FCA to automatically generate abstract features (super-concepts) and their partial ordering (the concept lattice), recording and transferring knowledge more generally.

Concept-Enabled AMDPs

A direct extension of both CAFE and AMDP hierarchies would be to unite them under the aegis of concept-enabled AMDPs. Both the generation of a hierarchy and the subsequent learning of the AMDP models could leverage CAFE, determining subtask parameters and relevant features. In terms of learning a hierarchy itself, an agent could be presented with a curriculum of increasingly complex, related tasks and learn concept-based subtasks incrementally to then combine into a task hierarchy. Concept-based task descriptors would

describe abstractions over actions, generating specific subtasks based on what grounded concepts exist that can satisfy its parameters. With the example parameterized subtask described by the concept of "a key matching a door," a concept-enabled AMDP would be proposed for each combination of key and door, allowing the agent to reason directly over the objects relevant to the abstract goal of unlocking doors. Furthermore, conceptually and functionally similar objects in a state could be aggregated into a compositional object. A hierarchy of concept-enabled AMDPs could then plan top-down over the creation of these conceptual, composite objects, to build complex structures. Concepts could be used to describe the template (constraints and relations) that such component objects must satisfy to be combined into a more abstract compositional object like a house. An agent could plan over wall-building subtasks, using concept-parameterized task descriptors to arrange blocks into walls that satisfy the concept of a room. Concepts, therefore, could have a multifaceted function, as mechanisms for abstractions of states, actions, and goals. Concept-based abstractions will be compared to existing methods such as hierarchies of AMDPs and state aggregation as used in MAXQ.

Bibliography

[1] David Abel. Concepts in bounded rationality: Perspectives in reinforcement learning. Master's thesis, Brown University, 2019.

[2] David Abel, D. Ellis Hershkowitz, Gabriel Barth-Maron, Stephen Brawner, Kevin O'Farrell, James MacGlashan, and Stefanie Tellex. Goal-based action priors. In *Proceedings of the 25th International Conference on Automated Planning and Scheduling*, pages 306–314, 2015.

[3] David Abel, Yuu Jinnai, Sophie Yue Guo, George Konidaris, and Michael Littman. Policy and value transfer in lifelong reinforcement learning. In *International Conference on Machine Learning*, pages 20–29, 2018.

[4] David Abel, John Winder, Marie desJardins, and Michael Littman. The expected-length model of options. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. AAAI Press, 2019.

[5] James S. Albus. A theory of cerebellar function. *Mathematical Biosciences*, 10(1-2):25–61, 1971.

[6] Ron Alford, Vikas Shivashankar, Mark Roberts, Jeremy Frank, and David W. Aha. Hierarchical planning: relating task and goal decomposition with task sharing. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 3022–3028. AAAI Press, 2016.

[7] Simon Andrews. IN-CLOSE2, a high performance formal concept miner. In *Proceedings of the International Conference on Conceptual Structures*, pages 50–62. Springer, 2011.

[8] Simon Andrews, Constantinos Orphanides, and Simon Polovina. Visualising computational intelligence through converting data into formal concepts. In *Next Generation Data Technologies for Collective Computational Intelligence*.

[9] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, 34(6):26–38, 2017.

[10] Dilip Arumugam, Siddharth Karamcheti, Nakul Gopalan, Edward C Williams, Mina Rhee, Lawson LS Wong, and Stefanie Tellex. Grounding natural language instructions to semantic goal representations for abstraction and generalization. *Autonomous Robots*, 43(2):449–468, 2019.

[11] Kavosh Asadi, Dipendra Misra, and Michael L Littman. Lipschitz continuity in model-based reinforcement learning. *ICML*, 2018.

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

[13] Aijun Bai, Siddharth Srivastava, and Stuart Russell. Markovian state and action abstractions for mdps via hierarchical mcts. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 3029–3037. AAAI Press, 2016.

[14] Aijun Bai, Feng Wu, and Xiaoping Chen. Online planning for large mdps with maxq decomposition. In *AAMAS*, 2012.

[15] Aijun Bai, Feng Wu, and Xiaoping Chen. Online planning for large Markov decision processes with hierarchical decomposition. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(4):45, 2015.

[16] Bram Bakker, Zoran Zivkovic, and Ben Krose. Hierarchical dynamic programming for robot path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005.

[17] Jennifer L. Barry, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Deth: Approximate hierarchical solution of large Markov decision processes. In *International Joint Conference on Artificial Intelligence*, 2011.

[18] Andrew G Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 13(1-2):41–77, 2003.

[19] Jiri Baum, Ann E Nicholson, and Trevor I Dix. Proximity-based non-uniform abstractions for approximate planning. *Journal of Artificial Intelligence Research*, 43:477–522, 2012.

[20] R. Bellman. A Markovian decision process. *Indiana University Mathematics Journal*, 6:679–684, 1957.

[21] Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. Weight uncertainty in neural networks. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pages 1613–1622. JMLR. org, 2015.

[22] Ronen I. Brafman and Moshe Tennenholtz. R-MAX: A general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 3(Oct):213–231, 2002.

[23] Emma Brunskill and Lihong Li. PAC-inspired option discovery in lifelong reinforcement learning. In *Proceedings of The 31st International Conference on Machine Learning*, pages 316–324, 2014.

[24] Martin V. Butz, Samarth Swarup, and David E. Goldberg. Effective online detection of task-independent landmarks. *Online Proceedings for the ICML'04 Workshop on Predictive Representations of World Knowledge*, 2004.

[25] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[26] L. C. Cobo, C. L. Isbell, and A. L. Thomaz. Object focused Q-learning for autonomous agents. In *Proceedings of the International Conference on Autonomous Agents and Multi-Agent Systems*, pages 1061–1068, 2013.

[27] Felipe Leno da Silva and Anna Helena Reali Costa. Accelerating multiagent reinforcement learning through transfer learning. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 5034–5035. AAAI Press, 2017.

[28] Felipe Leno Da Silva and Anna Helena Reali Costa. A survey on transfer learning for multiagent reinforcement learning systems. *Journal of Artificial Intelligence Research*, 64:645–703, 2019.

[29] Felipe Leno da Silva, Ruben Glatt, and Anna Helena Reali Costa. Object-oriented reinforcement learning in cooperative multiagent domains. In *5th Brazilian Conference on Intelligent Systems (BRACIS)*, pages 19–24. IEEE, 2016.

[30] Felipe Leno da Silva, Ruben Glatt, and Anna Helena Reali Costa. An advising framework for multiagent reinforcement learning systems. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI-17)*, pages 4913–4914. AAAI Press, 2017.

[31] Thomas Dean and Shieu-Hong Lin. Decomposition techniques for planning in stochastic domains. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence-Volume 2*, pages 1121–1127. Morgan Kaufmann Publishers Inc., 1995.

[32] Marie desJardins, Tenji Tembo, Nicholay Topin, Michael Bishoff, Shawn Squire, James MacGlashan, Rose Carignan, and Nicholas Haltmeyer. Discovering subgoals in complex domains. In *Working Notes of the AAAI 2014 Fall Symposium on Knowledge, Skill, and Behavior Transfer in Autonomous Robots*. AAAI Press, November 2014.

[33] Dua Dheeru and Efi Karra Taniskidou. UCI machine learning repository, 2017.

[34] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303, 2000.

[35] C. Diuk, M. L. Littman, and A.L. Strehl. A hierarchical approach to efficient reinforcement learning in deterministic domains. In *International Conference on Antonomous Agents and Multiagent Systems*, 2006.

[36] Carlos Diuk, Andre Cohen, and Michael L. Littman. An object-oriented representation for efficient reinforcement learning. In *Proceedings of the 25th International Conference on Machine Learning*, pages 240–247. ACM, 2008.

[37] Blerim Emruli and Fredrik Sandin. Analogical mapping with sparse distributed memory: A simple model that learns to generalize from examples. *Cognitive Computation*, 6(1):74–88, 2014.

[38] K. Erol, J. Hendler, and D.S. Nau. HTN planning: Complexity and expressivity. In *AAAI*, 1994.

[39] Kutluhan Erol. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation*. PhD thesis, College Park, MD, USA, 1996. UMI Order No. GAX96-22054.

[40] Richard E Fikes and Nils J Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[41] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135. JMLR. org, 2017.

[42] Douglas H. Fisher, Michael J. Pazzani, and Pat Langley, editors. *Concept Formation: Knowledge and Experience in Unsupervised Learning*. Morgan Kaufmann, 2014.

[43] Jerry Fodor. Concepts: A potboiler. *Cognition*, 50(1-3):95–113, 1994.

[44] Jerry A Fodor. *Concepts: Where Cognitive Science Went Wrong*. Oxford University Press, 1998.

[45] Gottlob Frege, PT Geach, and Max Black. On concept and object. *Mind*, 60(238):168–180. Translated 1951, original date 1892.

[46] Marta Garnelo, Kai Arulkumaran, and Murray Shanahan. Towards deep symbolic reinforcement learning. In *Deep Reinforcement Learning Workshop at the 30th Conference on Neural Information Processing Systems*, 2016.

[47] Marta Garnelo and Murray Shanahan. Reconciling deep learning with symbolic artificial intelligence: representing objects and relations. *Current Opinion in Behavioral Sciences*, 29:17–23, 2019.

[48] Matthieu Geist and Olivier Pietquin. Parametric value function approximation: A unified view. In *2011 IEEE Symposium on Adaptive Dynamic Programming And Reinforcement Learning (ADPRL)*, pages 9–16. IEEE, 2011.

[49] Matthieu Geist and Olivier Pietquin. Algorithmic survey of parametric value function approximation. *IEEE Transactions on Neural Networks and Learning Systems*, 24(6):845–867, 2013.

[50] Alborz Geramifard, Thomas J. Walsh, Nicholas Roy, and Jonathan P. How. Batch-iFDD for representation expansion in large MDPs. In *Uncertainty in Artificial Intelligence*, page 242, 2013.

[51] Alborz Geramifard, Thomas J. Walsh, Stefanie Tellex, Girish Chowdhary, Nicholas Roy, and Jonathan P. How. A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Foundations and Trends in Machine Learning*, 6(4):375–451, 2013.

[52] Behzad Ghazanfari and Nasser Mozayani. Extracting bottlenecks for reinforcement learning agent by holonic concept clustering and attentional functions. *Expert Systems with Applications*, 54:61–77, 2016.

[53] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: an approach to evaluating interpretability of machine learning. *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, 2018.

[54] Nakul Gopalan, Marie desJardins, Michael L. Littman, James MacGlashan, Shawn Squire, Stefanie Tellex, John Winder, and Lawson LS Wong. Planning with abstract Markov decision processes. In *The 33rd International Conference on Machine Learning Workshop on Abstraction in RL*, 2016.

[55] Nakul Gopalan, Marie desJardins, Michael L. Littman, James MacGlashan, Shawn Squire, Stefanie Tellex, John Winder, and Lawson LS Wong. Planning with abstract Markov decision processes. In *International Conference on Automated Planning and Scheduling*, 2017.

[56] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[57] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[58] Erik Harpstead, Christopher J MacLellan, Kenneth R Koedinger, Vincent Aleven, Steven P Dow, and Brad Myers. Investigating the solution space of an open-ended educational game using conceptual feature extraction. 2013.

[59] Anna Harutyunyan, Peter Vrancx, Pierre-Luc Bacon, Doina Precup, and Ann Nowé. Learning with options that terminate off-policy. In *AAAI*, 2018.

[60] Demis Hassabis. Artificial intelligence: Chess match of the century. *Nature*, 544(7651):413, 2017.

[61] Mohamad H. Hassoun, editor. *Associative Neural Memories: Theory and Implementation*. New York : Oxford University Press, 1993.

[62] Nicholas Hay, Michael Stark, Alexander Schlegel, Carter Wendelken, Dennis Park, Eric Purdy, Tom Silver, D Scott Phoenix, and Dileep George. Behavior is everything: Towards representing concepts with sensorimotor contingencies. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[63] Mikael Henaff, Jason Weston, Arthur Szlam, Antoine Bordes, and Yann LeCun. Tracking the world state with recurrent entity networks. *arXiv preprint arXiv:1612.03969*, 2016.

[64] Bernhard Hengst. *Hierarchical Reinforcement Learning*, pages 611–619. Springer US, Boston, MA, 2017.

[65] David Hume. Enquiry Concerning Human Understanding. 1748.

[66] Ulit Jaidee and Héctor Muñoz-Avila. CLASS Q-L: A Q-learning algorithm for adversarial real-time strategy games. In *Working Notes of the AIIDE Workshop on Adversarial Real-Time Games*, 2015.

[67] William James. *Habit*. H. Holt, 1890.

[68] Nicholas K Jong, Todd Hester, and Peter Stone. The utility of temporal abstraction in reinforcement learning. In *AAMAS*, pages 299–306, 2008.

[69] Nicholas K. Jong and Peter Stone. Hierarchical model-based reinforcement learning: R-MAX+ MAXQ. In *Proceedings of the 25th International Conference on Machine Learning*, pages 432–439, 2008.

[70] Anders Jonsson and Andrew Barto. A causal approach to hierarchical decomposition of factored MDPs. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 401–408. ACM, 2005.

[71] Leslie Pack Kaelbling and Tomás Lozano-Pérez. Hierarchical task and motion planning in the now. In *2011 IEEE International Conference on Robotics and Automation*, pages 1470–1477. IEEE, 2011.

[72] Sham M Kakade, Shai Shalev-Shwartz, and Ambuj Tewari. Efficient bandit algorithms for online multiclass prediction. In *Proceedings of the 25th International Conference on Machine Learning*, pages 440–447. ACM, 2008.

[73] Pentti Kanerva. Sparse distributed memory and related models. In *Associative Neural Memories*, pages 50–76. Oxford University Press, Inc., 1993.

[74] Siddharth Karamcheti. Grounding natural language to goals for abstraction, generalization, and interpretability. Master's thesis, Brown University, 2018.

[75] Siddharth Karamcheti, Edward C Williams, Dilip Arumugam, Mina Rhee, Nakul Gopalan, Lawson LS Wong, and Stefanie Tellex. A tale of two draggns: A hybrid approach for interpreting action-oriented and goal-oriented instructions. *ACL 2017*, page 67, 2017.

[76] Michael Kearns and Satinder Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 2002.

[77] Anthony Kenny. Concepts, brains, and behaviour. *Grazer Philosophische Studien*, 81(1), 2010.

[78] David Kent, Siddhartha Banerjee, and Sonia Chernova. Learning sequential decision tasks for robot manipulation with abstract Markov decision processes and demonstration-guided exploration. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 1–8. IEEE, 2018.

[79] Craig A Knoblock. An analysis of abstrips. In *Artificial Intelligence Planning Systems*, pages 126–135. Elsevier, 1992.

[80] George Konidaris and Andrew G. Barto. Building portable options: Skill transfer in reinforcement learning. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 895–900, 2007.

[81] George Konidaris and Andrew G Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *NeurIPS*, pages 1015–1023, 2009.

[82] George Konidaris, Leslie Pack Kaelbling, and Tomas Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61:215–289, 2018.

[83] George Konidaris, Scott Kuindersma, Roderic Grupen, and Andrew Barto. Robot learning from demonstration by constructing skill trees. *The International Journal of Robotics Research*, 31(3):360–375, 2012.

[84] George Konidaris, Sarah Osentoski, and Philip Thomas. Value function approximation in reinforcement learning using the Fourier basis. In *Proceedings of the 25th AAAI Conference on Artificial Intelligence*, pages 380–385. AAAI Press, 2011.

[85] George Konidaris, Ilya Scheidwasser, and Andrew Barto. Transfer in reinforcement learning via shared features. *Journal of Machine Learning Research*, 13(May):1333–1371, 2012.

[86] Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[87] Tor Lattimore and Csaba Szepesvari. *Bandit Algorithms*. Cambridge University Press, 2019.

[88] Miguel Lázaro-Gredilla, Dianhuan Lin, J Swaroop Guntupalli, and Dileep George. Beyond imitation: Zero-shot task transfer on robots by learning concepts as cognitive programs. *arXiv preprint arXiv:1812.02788*, 2018.

[89] Lihong Li. Sample complexity bounds of exploration. In *Reinforcement Learning*, pages 175–204. Springer, 2012.

[90] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*, pages 661–670. ACM, 2010.

[91] Lihong Li, Michael L Littman, Thomas J Walsh, and Alexander L Strehl. Knows what it knows: a framework for self-aware learning. *Machine learning*, 82(3):399–443, 2011.

[92] Lihong Li, Thomas J Walsh, and Michael L Littman. Towards a unified theory of state abstraction for mdps. In *In Proceedings of the Ninth International Symposium on Artificial Intelligence and Mathematics*. Citeseer, 2006.

[93] Christian Lindig. Fast concept analysis. In *Working with Conceptual Structures – Contributions to ICCS 2000*, pages 152–161. Shaker Verlag, 2000.

[94] Michael L Littman, Thomas L Dean, and Leslie Pack Kaelbling. On the complexity of solving Markov decision problems. In *UAI*, pages 394–402. Morgan Kaufmann Publishers Inc., 1995.

[95] J. MacGlashan, M. Babeş-Vroman, M. desJardins, M. L. Littman, S. Muresan, S. Squire, S. Tellex, D. Arumugam, and L. Yang. Grounding English commands to reward functions. In *Robotics: Science and Systems*, 2015.

[96] James MacGlashan. *Multi-Source Option-Based Policy Transfer*. PhD thesis, University of Maryland, Baltimore County, 2013.

[97] James MacGlashan. Burlap: Brown-UMBC Reinforcement Learning and Planning. http://burlap.cs.brown.edu/, 2016.

[98] Marlos C Machado, Marc G Bellemare, and Michael Bowling. A Laplacian framework for option discovery in reinforcement learning. *ICML*, 2017.

[99] Christopher J MacLellan. *Computational Models of Human Learning: Applications for Tutor Development and Theory Testing*. PhD thesis, Carnegie Mellon University Pittsburgh, PA, 2016.

[100] Daniel J Mankowitz, Timothy A Mann, and Shie Mannor. Time regularized interrupting options. In *ICML*, 2014.

[101] Timothy Mann and Shie Mannor. Scaling up approximate value iteration with options: Better policies with fewer iterations. In *ICML*, 2014.

[102] Timothy A Mann and Shie Mannor. The advantage of planning with options. *RLDM 2013*, page 9, 2013.

[103] Shie Mannor, Ishai Menache, Amit Hoze, and Uri Klein. Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the 21st International Conference on Machine Learning*, 2004.

[104] Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the 18th International Conference on Machine Learning*, pages 361–368, 2001.

[105] H Brendan McMahan, Maxim Likhachev, and Geoffrey J Gordon. Bounded real-time dynamic programming: RTDP with monotone upper bounds and performance guarantees. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 569–576. ACM, 2005.

[106] D. L. Medin and E. E. Smith. Concepts and concept formation. *Annual Review of Psychology*, 35:113, 1984.

[107] Neville Mehta. *Hierarchical Structure Discovery and Transfer in Sequential Decision Problems*. PhD thesis, Oregon State University, 2011.

[108] Neville Mehta, Soumya Ray, Prasad Tadepalli, and Thomas Dietterich. Automatic discovery and transfer of MAXQ hierarchies. In *Proceedings of the 25th International Conference on Machine Learning*, pages 648–655, 2008.

[109] Francisco S. Melo, Sean P. Meyn, and M. Isabel Ribeiro. An analysis of reinforcement learning with function approximation. In *Proceedings of the 25th International Conference on Machine Learning*, pages 664–671. ACM, 2008.

[110] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut: Dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the 13th European Conference on Machine Learning*, pages 295–306. Springer, 2002.

[111] John Menick. Move 37: Artificial intelligence, randomness, and creativity. *Mousse Magazine*, 53, 2016.

[112] Tom M Mitchell. Version spaces: A candidate elimination approach to rule learning. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, pages 305–310. Morgan Kaufmann Publishers Inc., 1977.

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

[114] Igor Mordatch. Concept learning with energy-based models. In *ICLR 2018 Workshop*, 11 2018.

[115] A Mousavi, M Nili Ahmadabadi, H Vosoughpour, BN Araabi, and N Zaare. Hierarchical functional concepts for knowledge transfer among reinforcement learning agents. *Iranian Journal of Fuzzy Systems*, 12(5):99–116, 2015.

[116] Negin Nejati, Pat Langley, and Tolga Konik. Learning hierarchical task networks by observation. In *Proceedings of the 23rd International Conference on Machine Learning*, 2006.

[117] Ronald Ortner. Adaptive aggregation for reinforcement learning in average reward Markov decision processes. *Annals of Operations Research*, 208(1):321–336, 2013.

[118] R. Parr and S. Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems: Proceedings of the 1997 Conference*, 1998.

[119] Ron Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In *Advances in Neural Information Processing Systems 11*, pages 1043–1049, 1998.

[120] Ronald Parr. Flexible decomposition algorithms for weakly coupled Markov decision problems. In *Proceedings of the 14th conference on Uncertainty in artificial intelligence*, pages 422–430. Morgan Kaufmann Publishers Inc., 1998.

[121] Marc Pickett and Andrew G. Barto. PolicyBlocks: An algorithm for creating useful macro-actions in reinforcement learning. In *Proceedings of the 19th International Conference on Machine Learning*, pages 506–513, 2002.

[122] János Podani. *Introduction to the Exploration of Multivariate Biological Data*. Backhuys Publishers, 2000.

[123] Jonas Poelmans, Dmitry I. Ignatov, Sergei O. Kuznetsov, and Guido Dedene. Formal concept analysis in knowledge processing: A survey on applications. *Expert Systeme with Applications*, 40(16):6538–6560, 2013.

[124] Jonas Poelmans, Sergei O. Kuznetsov, Dmitry I. Ignatov, and Guido Dedene. Formal concept analysis in knowledge processing: A survey on models and techniques. *Expert Systems with Applications*, 40(16):6601–6623, 2013.

[125] Marc Ponsen, Matthew E Taylor, and Karl Tuyls. Abstraction and generalization in reinforcement learning: A summary and framework. In *International Workshop on Adaptive and Learning Agents*, pages 1–32. Springer, 2009.

[126] Doina Precup and Richard S Sutton. Multi-time models for reinforcement learning. In *ICML Workshop on Modelling in Reinforcement Learning*, 1997.

[127] Doina Precup and Richard S Sutton. Multi-time models for temporally abstract planning. In *NeurIPS*, pages 1050–1056, 1998.

[128] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[129] Ramya Ramakrishnan, Karthik Narasimhan, and Julie Shah. Interpretable transfer for reinforcement learning based on object similarities. In *Proceedings of the IJCAI Workshop on Interactive Machine Learning*, 2016.

[130] Carlos Riquelme, George Tucker, and Jasper Snoek. Deep Bayesian bandits showdown: An empirical comparison of bayesian deep networks for thompson sampling. *6th International Conference on Learning Representations, ICLR 2018*, 2018.

[131] Melrose Roderick, Christopher Grimm, and Stefanie Tellex. Deep abstract q-networks. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 131–138. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

[132] G. A. Rummery and M. Niranjan. On-line Q-learning using connectionist systems. Technical Report TR 166, Cambridge University Engineering Department, Cambridge, England, 1994.

[133] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (3rd Edition)*. Prentice Hall, 3 edition, 2009.

[134] Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on Thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

[135] Earl D Sacerdoti. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135, 1974.

[136] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. One-shot learning with memory-augmented neural networks. *arXiv preprint arXiv:1605.06065*, 2016.

[137] Jeff Schlimmer. Mushroom records drawn from the Audubon Society field guide to North American mushrooms. *GH Lincoff (Pres), New York*, 1981.

[138] Jürgen Schmidhuber. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

[139] D Silver and K Ciosek. Compositional planning using optimal option models. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012*, volume 2, pages 1063–1070, 2012.

[140] David Silver, Aja Huang, Chris J. Maddison, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[141] Özgür Şimşek and A. Barto. Betweenness centrality as a basis for forming skills. Technical Report TR-2007-26, University of Massachusetts Department of Computer Science, 2007.

[142] Özgür Şimşek and Andrew G Barto. Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the 21st International Conference on Machine Learning*, page 95. ACM, 2004.

[143] Özgür Şimşek, Alicia P. Wolfe, and Andrew G. Barto. Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 816–823, 2005.

[144] Satinder P Singh, Andrew G Barto, and Nuttapong Chentanez. Intrinsically motivated reinforcement learning. In *NeurIPS*, pages 1281–1288, 2005.

[145] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation*, pages 212–223, London, UK, UK, 2002. Springer-Verlag.

[146] Alexander L Strehl. Probably approximately correct (PAC) exploration in reinforcement learning. 2007.

[147] Masashi Sugiyama. *Statistical reinforcement learning: modern machine learning approaches*. Chapman and Hall/CRC, 2015.

[148] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 1st edition, 1998.

[149] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

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

[151] István Szita and Csaba Szepesvári. Agnostic KWIK learning and efficient approximate reinforcement learning. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 739–772, 2011.

[152] Erik Talvitie. Self-correcting models for model-based reinforcement learning. In *AAAI*, 2017.

[153] Matthew E Taylor and Peter Stone. Behavior transfer for value-function-based reinforcement learning. In *Proceedings of the 4th international joint conference on Autonomous agents and multiagent systems*, pages 53–59. ACM, 2005.

[154] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(Jul):1633–1685, 2009.

[155] Matthew E Taylor and Peter Stone. An introduction to intertask transfer for reinforcement learning. *AI Magazine*, 32(1):15, 2011.

[156] Ana C Tenorio-González and Eduardo F Morales. Automatic discovery of relational concepts by an incremental graph-based representation. *Robotics and Autonomous Systems*, 83:1–14, 2016.

[157] Gerald Tesauro. Temporal difference learning and TD-gammon. *Communications of the ACM*, 38:58–68, March 1995.

[158] William R Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3/4):285–294, 1933.

[159] Nicholay Topin, Nicholas Haltmeyer, Shawn Squire, John Winder, Marie desJardins, and James MacGlashan. Portable option discovery for automated learning transfer in object-oriented Markov decision processes. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 3856–3864. AAAI Press, 2015.

[160] Paul Tseng. Solving h-horizon, stationary Markov decision problems in time proportional to log (h). *Operations Research Letters*, 9(5):287–297, 1990.

[161] John N Tsitsiklis and Benjamin Van Roy. Analysis of temporal-difference learning with function approximation. In *Advances in Neural Information Processing Systems*, pages 1075–1081, 1997.

[162] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, October 1950. ArticleType: primary_article / Full publication date: Oct., 1950 / Copyright © 1950 Oxford University Press.

[163] Renee Tursi. William james's narrative of habit. *Style*, 33(1):67–87, 1999.

[164] Rosana Veroneze, Arindam Banerjee, and Fernando J. Von Zuben. Enumerating all maximal biclusters in real-valued datasets. *Information Sciences*, 379:288–309, February 2017.

[165] Christopher M Vigorito and Andrew G Barto. Intrinsically motivated hierarchical skill learning in structured environments. *IEEE Transactions on Autonomous Mental Development*, 2(2):132–143, 2010.

[166] Thomas J Walsh, István Szita, Carlos Diuk, and Michael L Littman. Exploring compact reinforcement-learning representations with linear regression. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence*, pages 591–598. AUAI Press, 2009.

[167] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[168] Martha White. Unifying task specification in reinforcement learning. In *ICML*, pages 3742–3750, 2017.

[169] Shimon Whiteson. Adaptive tile coding. In *Adaptive Representations for Reinforcement Learning*, pages 65–76. Springer, 2010.

[170] John Winder and Marie desJardins. Concept-aware feature extraction for knowledge transfer in reinforcement learning. In *KEG-18: 2018 AAAI Workshop on Knowledge Extraction from Games*, 2018.

[171] John Winder, Shawn Squire, Matthew Landen, Stephanie Milani, and Marie desJardins. Towards planning with hierarchies of learned Markov decision processes. In *ICAPS Integrated Execution of Planning and Acting (IntEx) Workshop*, 2017.

[172] Cheng Wu. *Novel Function Approximation Techniques for Large-scale Reinforcement Learning*. PhD thesis, Northeastern University, 2010.

[173] Angela Yang and Grace Hui Yang. A contextual bandit approach to dynamic search. In *Proceedings of the ACM SIGIR International Conference on Theory of Information Retrieval*, pages 301–304. ACM, 2017.

[174] Edward N. Zalta. Gottlob Frege. In Edward. N. Zalta, editor, *The Stanford encyclopedia of philosophy*. Summer 2018 ed. edition, 2018.

[175] Bin Zhang and Sargur N Srihari. Properties of binary vector dissimilarity measures. In *Proceedings of JCIS International Conference on Computer Vision, Pattern Recognition and Image Processing*, volume 1. ACM, 2003.

[176] Li Zhou and Emma Brunskill. Latent contextual bandits and their application to personalized recommendations for new users. In *Proceedings of the 25th International Joint Conference on Artificial Intelligence*, pages 3646–3653. AAAI Press, 2016.