BEHAVIORAL BUILDING BLOCKS FOR AUTONOMOUS AGENTS: DESCRIPTION, IDENTIFICATION, AND LEARNING

A Dissertation Presented

by

ÖZGÜR ŞİMŞEK

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2008

Department of Computer Science

BEHAVIORAL BUILDING BLOCKS FOR AUTONOMOUS AGENTS: DESCRIPTION, IDENTIFICATION, AND LEARNING

A Dissertation Presented

by

ÖZGÜR ŞİMŞEK

Approved as to style and content by:

Andrew G. Barto, Chair

David Jensen, Member

Michael Littman, Member

Sridhar Mahadevan, Member

Andrea R. Nahmod, Member

Andrew G. Barto, Department Chair
Department of Computer Science

ACKNOWLEDGMENTS

It has been a joy to have Andy Barto as my thesis advisor. I would like to thank him in particular for encouraging my ideas from the early stages of their conception and for his support during the long and difficult time I spent in Turkey, waiting for the renewal of my visa to the United States.

Thanks to Alicia P. Wolfe for our collaboration on developing a skill-discovery algorithm based on local graph partitioning. It was great fun and her expertise in graph partitioning informed key components of the algorithm.

Thanks to Konstantinos Katsikopoulos for suggesting that I take a look at *Reinforcement Learning: An Introduction* by Rich Sutton and Andy Barto, which was one of the early steps that led to this dissertation.

Thanks to David Jensen, Michael Littman, Sridhar Mahadevan, Rich Sutton, Doina Precup, Manuela Veloso, Andrew McCallum, and Andrea Nahmod for their feedback on my research at its various stages.

Thanks to all past and present members of the Autonomous Learning Laboratory, the Knowledge Discovery Laboratory, and the Information Extraction and Synthesis Laboratory at UMass. Thanks especially to Dan Bernstein, Lisa Friedland, Anders Jonsson, Mohammad Ghavamzadeh, George Konidaris, Victoria Manfredi, Amy McGovern, Jennifer Neville, Balaraman Ravindran, Matt Rattigan, Khashayar Rohanimanesh, Ashwin Shah, Agustin Schapira, and Alicia P. Wolfe, who I spent extensive time discussing ideas.

Thanks to Alicia P. Wolfe and Jennifer Neville for lending their ears whenever I needed them.

Thanks to Andy Barto, David Jensen, Andrew McCallum, UMass Amherst Graduate School, and the UMass Amherst Isenberg School of Management for their financial support in graduate school.

And finally, thanks to my parents, Sevil and Hüseyin Şimşek, my sister, Özlem Şimşek-Kiper, my brother, Onur Şimşek, my husband, Konstantinos Katsikopoulos, and all of my dear friends in Amherst who have made my time here such a bliss.

ABSTRACT

BEHAVIORAL BUILDING BLOCKS FOR AUTONOMOUS AGENTS: DESCRIPTION, IDENTIFICATION, AND LEARNING

SEPTEMBER 2008

ÖZGÜR ŞİMŞEK

B.S., BOĞAZIÇI ÜNIVERSITESI

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

M.S., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew G. Barto

The broad problem I address in this dissertation is design of autonomous agents that can efficiently learn how to achieve desired behaviors in large, complex environments. I focus on one essential design component: the ability to form new behavioral units, or *skills*, from existing ones. I propose a characterization of a useful class of skills in terms of general properties of an agent's interaction with its environment—in contrast to specific properties of a particular environment—and I introduce methods that can be used to identify and acquire such skills autonomously.

CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

The broad problem I address in this dissertation is design of autonomous agents that are able to efficiently learn how to achieve desired outcomes in large, complex environments. I focus on one essential design component: the ability to autonomously form useful skills. In the following sections, I describe what I mean by a skill, explain why skills are useful, and outline the contents of this dissertation.

1.1 What Is a Skill?

A *skill* is a behavior that an agent may exhibit in its environment. It is composed of the actions that are made available to the agent by the system designer and it may itself be treated as a single action when learning, planning, or determining how to act. Grasping, driving, and walking are examples of skills we use frequently. For a mobile robot, we can define a skill for moving to a particular location. A strategy that directs the individual moves of a chess player is also a skill.

The daily use of "skill" implies doing something well, but this meaning of the word is not implied in its use here. A skill, as defined here, may specify any behavior—although a skill would not be particularly useful unless it does something well.

The skills I consider in this dissertation are composed of discrete actions only. They do not include skills defined with continuous actions such as grasping. The ideas presented here, however, may be extended in the future to be applicable in the continuous case.

1.2 Why Are Skills Useful?

Skills do not augment the set of behaviors that an agent may exhibit. Those are determined by the actions that are built in by the system designer. Skills simply define a set of behaviors that the agent may treat as individual behavioral units. What, then, can be gained by forming skills?

The answer is speed. Problem solving proceeds by trying different actions and evaluating their consequences. Built-in actions typically are too fine units of behavior to be used in this procedure, to the extent that they often do not lead to a solution, optimal or not, in a reasonable amount of time. In contrast, a suitable set of skills specifying coarser units of behavior may make it possible to identify a solution with ease. The trade-off is that a given skill set may be unable to represent all possible solutions and therefore may lead to a solution inferior to the one theoretically possible with primitive actions. This trade-off is often worth it, and is often necessary, because it makes it possible to find a satisfactory solution.

An additional incentive is present for forming skills if the agent is to face a sequence of related problems: skills formed while solving one problem may be useful for solving subsequent problems. The ability to grasp objects, for example, is useful for solving many problems that a robot may face. Some degree of transfer is still possible even if the new environment is not entirely the same as the environment in which the skill was acquired. If objects have a different shape than those experienced before, the skill for grasping objects may be adapted to the new shape while higher-level skills that use grasping remain usable as they are.

Perhaps the most important reason for equipping artificial agents with the ability to form useful skills is to enable an autonomous developmental process. Equipped with this ability, an agent can learn to display behaviors of increasing complexity through continuously building on its existing skills to acquire new ones. For example, grasping may be followed by manipulating objects in different ways, which may be

followed by using a key to unlock the door to an adjacent room, and so on, forming a continuously growing skill hierarchy. In this process, the agent develops the capability to perform tasks of increasing difficulty without guidance tailored for the specific environment it operates in. Instead, the capabilities developed are the result of a generic developmental process that, in a different environment, would result in learning to perform an entirely different set of tasks. This type of open-ended developmental process is fundamentally different from how artificial agents learn to perform complex tasks today—by considerable human design effort tailored for specific tasks—and has the potential to dramatically increase the capabilities of artificial agents. The argument for this type of open-ended developmental learning has been made convincingly in the literature by several researchers (e.g., Barto et al. 2004; Weng et al. 2001).

1.3 Outline

In this dissertation, I address a sequence of questions that are of fundamental importance for equipping artificial agents with the ability to form useful skills on their own.

1. *What constitutes a useful skill? In other words, what are the fundamental properties of skills that make them useful, as useful, for example, as grasping is for agents that routinely manipulate objects? More importantly, can we express the answer without reference to particular properties of a specific task or a domain, using instead properties that are shared across different tasks and domains?*

   I characterize a set of skills using a graphical representation of an agent's interaction with its environment. This characterization uses *betweenness*, a measure of centrality on graphs, to capture a set of skills that allows efficient navigation on the interaction graph by exploiting its structural properties. Because these skills allow efficient access to different regions of the graph, I call them *access*

skills. I show that this single concept captures a wide range of skills in a diverse set of environments. These skills are consistent with common sense, are similar to skills that people handcraft for these domains, and improve learning performance. In the game of Tic-Tac-Toe, access skills set up a fork, forcing the opponent to lose. In the Towers of Hanoi puzzle, access skills include clearing the stack above the largest disk and clearing another peg entirely to be able to move the largest disk to another peg.

2. *How can an agent identify such skills?*

   The definition of access skills may be used directly as a discovery algorithm to form a set of skills suitable for a given environment. I also develop three low-cost, incremental algorithms that do not require explicit or complete representation of the interaction graph. Instead, these algorithms use short paths sampled from the graph while the agent interacts with its environment.

3. *How can an agent efficiently acquire a desired skill? For instance, how can an agent efficiently and fully learn how to grasp, once it decides to acquire this skill?*

   While the previous question is concerned with identifying what the skill should accomplish, this third question is concerned with how to efficiently develop the skill. I formulate the optimization problem that the agent needs to address in this context and propose an algorithm for its approximate solution. The algorithm I introduce is not limited to acquiring access skills, but may be used to acquire a much broader set of skills: those that may be specified by a reward function they should maximize.

In Chapter 1, I provide a brief background on the learning framework I use in this dissertation and discuss the related work in the literature. In Chapters 3, 4, and

5, I address the three questions outlined above. I conclude with a discussion of the contributions of this dissertation.

CHAPTER 2

BACKGROUND AND RELATED WORK

The main assumption I make in this dissertation is that the interaction of the agent with its environment can be modeled as a Markov Decision Process (MDP). The MDP framework has proved over the years to be a fruitful formalism for addressing sequential decision-making problems. In this framework, the agent's interaction with its environment proceeds as follows. At each decision stage, the agent takes some action, observes the change in environment state, and receives a numerical reward signal. The agent's objective is to maximize some long-term measure of the reward it receives. I pay particular attention to the setting in which the MDP is not known, but needs to be discovered through interaction with the environment.

In the following sections, I review the basic concepts from the literature on MDPs and reinforcement learning that are directly relevant for this dissertation. For an in-depth introduction to these topics, the reader is referred to Howard (1960), Puterman (1994), Kaelbling et al. (1996), Bertsekas and Tsitsiklis (1996), and Sutton and Barto (1998).

2.1 Markov Decision Processes

A finite MDP is a tuple $\langle S, A, T, R, D, \gamma \rangle$, where S is a finite set of states, A is a finite state of actions, $T : S \times A \times S \rightarrow [0, 1]$ is a transition function, $R : S \times A \times S \rightarrow \Re$ is a reward function, $D : S \rightarrow [0, 1]$ is the initial state distribution from which the start state is drawn, and γ is a discount factor, $0 \leq \gamma \leq 1$. At each decision stage, the agent observes a state $s \in S$ and executes an action $a \in A$ with probability $\pi(s, a)$,

where $\pi : S \times A \rightarrow [0, 1]$ is a stationary stochastic policy. With probability $T(s, a, s')$, the agent observes state s' in the next decision stage and receives an immediate reward with expected value $R(s, a, s')$.

The objective is to maximize *return*, which is defined here as the discounted sum of future rewards. The value function of policy π is a map $V^\pi : S \rightarrow \Re$ that specifies the expected return for executing π starting from state s. An optimal policy is one that maximizes the value function over all states. The actual value of state s is distinct from the agent's estimate of it. To refer to the latter at decision stage t, I use $V_t(s)$. I use $V(\pi)$ to denote *policy value* with respect to the initial state distribution:

$$V(\pi) = \sum_{s \in S} D(s) V^\pi(s). \tag{2.1}$$

The action-value function of policy π is a map $Q^\pi : S \times A \rightarrow \Re$ that specifies the expected return from state s if the agent executes action a and thereafter follows policy π.

The *state-transition graph* of an MDP, or its *interaction graph*, is a weighted, directed graph in which the vertices correspond to the states in the MDP and the edges correspond to possible state transitions. A directed edge $v_1 \rightarrow v_2$ is present in the graph if and only if the corresponding state transition is possible through at least one action. Edge weights are equal to the reward expected when the corresponding transition takes place.

2.2 Options

An option represents a skill by specifying a policy to be followed during the skill's execution, the set of states at which the option is available as an action choice, and the probability of termination at each state in the domain. Specifically, an *option* is a triple $\langle I, \pi, \beta \rangle$, where I denotes the option's initiation set, i.e., the set of states in

which the option can be invoked, π denotes the policy followed when the option is executing, and $\beta : I \rightarrow [0, 1]$, denotes the option's termination condition, with $\beta(s)$ giving the probability that the option terminates in state $s \in I$ (Precup 2000; Sutton et al. 1999).

The work presented here is concerned with the behavior that a skill should exhibit. As such, the contributions of this dissertation are not tied to a particular representation. Although I represent skills as *options*, other representations may be used instead. Existing representations that are most similar to options are hierarchies of abstract machines (Parr and Russell 1998; Parr 1998) and the MAX-Q framework (Dietterich 2000).

2.3 Q-Learning

A simple and elegant algorithm for learning an optimal policy for an MDP is Q-learning (Watkins 1989), which may be applied without knowledge of the underlying MDP while the agent is interacting with its environment. The algorithm maintains an action-value function and updates it at every decision stage. The update equation after executing primitive action a in state s, observing next state s', and receiving reward r is

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

where α is a step-size parameter, $0 < \alpha \leq 1$.

If the agent's action repertoire includes options, the analogous update after initiating option o at state s, executing it for k steps until its termination at state s' is

$$Q(s, o) \leftarrow Q(s, o) + \alpha[\sum_{t=1}^{k} \gamma^{t-1} r_t + \gamma^k \max_{a' \in O} Q(s', a') - Q(s, o)],$$

where O is the set of all actions (both primitives and options) and $r_1..r_k$ are the sequence of rewards received during the option's execution. This equation is a Macro-Q learning update (McGovern et al. 1997), the discrete-time version of the Q-learning update in Semi-Markov Decision Processes (Bradke and Duff 1995).

The option values may also be updated while the skill is executing. After the agent executes primitive action a (through executing option o) at state s, receives reward r, and observes next state s', the update equation is

$$Q(s_t, o) \leftarrow Q(s_t, o) + \alpha[r + \gamma U(s_{t+1,o}) - Q(s, o)]$$

where

$$U(s, o) = (1 - \beta(s))Q(s, o) + \beta(s)\max_{o' \in O} Q(s, o').$$

This equation denotes one-step intra-option Q-learning update (Precup 2000). It may be applied not only to the option currently executing, but to every other option that would execute primitive action a with probability 1 at state s.

2.4 Domains and Tasks

To examine a skill's utility in addressing different problems in the same environment, it is useful to define an MDP in terms of a domain and a task.

Definition A *domain* is a tuple $\langle S, A, T_{\mathcal{D}}, R_{\mathcal{D}} \rangle$, where S is a finite set of states, A is a finite state of actions, $T_{\mathcal{D}} : S \times A \times S \rightarrow [0, 1]$ is the domain transition function and $R_{\mathcal{D}} : S \times A \times S \rightarrow \Re$ is the domain reward function.

Definition A *task* defined on a domain $\mathcal{D} = \langle S, A, T_{\mathcal{D}}, R_{\mathcal{D}} \rangle$ is a tuple $\langle G, R_{\mathcal{T}}, D, \gamma \rangle$, where $G \in S$ is a finite set of terminal states, $R_{\mathcal{T}} : S \times A \times S \rightarrow [0, 1]$ is the task reward function, $D : S \rightarrow [0, 1]$ is the initial state distribution from which the start state is drawn, and γ is a discount factor, $0 \leq \gamma \leq 1$.

A domain $\mathcal{D} = \langle S, A, T_{\mathcal{D}}, R_{\mathcal{D}} \rangle$, together with a task $\mathcal{T} = \langle G, R_{\mathcal{T}}, D, \gamma \rangle$ defined on that domain, specifies an MDP $\langle S, A, T, R, D, \gamma \rangle$, where

$$R(s,a) = \begin{cases} R_{\mathcal{D}}(s,a) + R_{\mathcal{T}}(s,a), & s \notin G \\ 0, & s \in G, \end{cases}$$

$$T(s,a,s') = \begin{cases} T_{\mathcal{D}}(s,a,s'), & s \notin G \\ 1, & s \in G, s = s' \\ 0, & s \in G, s \neq s'. \end{cases}$$

Definition The *domain interaction graph*, or the *domain state-transition graph*, is the interaction graph of the MDP specified by the domain and a task that has no terminal states and that assigns zero reward to all transitions.

2.5 Related Work

The utility of forming new behavioral units from existing ones has been recognized early in the Artificial Intelligence (AI) literature. Work on deterministic search problems introduced the notion of a *macro-operator*, or a *macro*, which is a sequence of primitive operators treated as a single operator (Amarel 1968). Subsequent work on stochastic problems has employed generalizations of this idea, forming closed-loop policies over the primitives whose execution depends on feedback from the environment. Some examples are *behaviors* (Brooks 1986), *activities* (Harel 1987), and *options* (Sutton et al. 1999). Here, I adopt the terminology of Thrun and Schwartz (1995) and call them *skills*.

Despite early recognition of the utility of skills in the AI literature, methods for forming skills autonomously have received relatively little attention, perhaps due to the difficulty of specifying methods that are independent of the specific context. Below, I discuss the main ideas that have been proposed for forming skills.

String together primitives that can be executed consecutively. For example, in addressing deterministic search problems, Dawson and Siklossy (1977) created macro-operators composed of successive execution of two primitive operators such that the post-conditions of the first operator matched the pre-conditions of the second one.

Form skills that jump over a "valley" between two "peaks", with respect to a heuristic evaluation function, if such a function is being used to guide the search. This principle was used by Iba (1989) to address deterministic search problems.

Find common action sequences in successful solutions to a single problem or a number of related problems. This principle was used by Iba (1989) to filter macro-actions discovered by other means and by McGovern (2002) as the basis for forming them. Related methods have been proposed by Thrun and Schwartz (1995) and by Pickett and Barto (2002).

Decompose the problem into smaller, easier problems and define skills for solving each of these. Korf (1985) has used macro-actions that achieve a subgoal of the problem without undoing subgoals already achieved. Methods more recently proposed by Hengst (2002), Jonsson and Barto (2005), Mehta et al. (2008), and Marthi et al. (2007) address problems in which state is specified by a set of variables. The algorithm by Hengst (2002) decomposes the problem into smaller tasks based on an ordering of the state variables with respect to their frequency of change. Jonsson and Barto (2005) and Mehta et al. (2008) decompose the problem based on causal relationships among the state variables. Marthi et al. (2007) search the space of all possible decompositions that are consistent with observed trajectories to identify the decomposition that minimizes a measure of learning time in future problems. In all of these algorithms, the subtasks

are substantially easier to solve than the main problem because irrelevant state variables are ignored when solving the subtasks.

Identify subgoals and form skills that efficiently take the agent to these states. Subgoals are states that are considered useful to reach for various reasons. Subgoals proposed in the literature include states that have a high reward gradient (Digney 1998), states that are visited frequently (Digney 1998; Stolle and Precup 2002; Stolle 2004), states that are visited frequently on successful trajectories but not on unsuccessful ones (McGovern and Barto 2001), states that are difficult to reach but easy to leave under a random walk (Bonarini et al. 2006), states that lie between densely-connected regions of the state space (Menache et al. 2002; Mannor et al. 2004), and states that lie between metastable regions of the state space (Mathew 2008), where a metastable region is informally defined as a region of the state space such that the time spent in the region is much larger than the time taken to transition to other metastable regions. Also in this category are the *narrows* of Amarel (1968), which he defines as "points of transition between easily traversible areas".

Access skills, the skills that I introduce in this dissertation, belong to the last category. They efficiently take the agent to certain states in the environment. They are closely related to the skills identified by the discovery algorithms of Stolle and Precup (2002), Stolle (2004), McGovern and Barto (2001), Menache et al. (2002), Mannor et al. (2004), and Mathew (2008). These algorithms are motivated by the "bottleneck" concept, which access skills capture and generalize. Consequently, there is an overlap in the skills that are identified by these existing algorithms and the skills introduced in this dissertation. This is a small overlap, with access skills encompassing a much larger set of skills than those identified by the existing algorithms. The algorithms by Stolle and Precup (2002) and Stolle (2004) are exceptions. The overlap with these algorithms is large, but the skill-discovery algorithms that are introduced

in this dissertation require drastically less computation and experience in the domain than these earlier algorithms.

Access skills are defined on a graphical representation of the agent's interaction with its environment. I use the structural properties of the interaction graph as a common language between different domains and tasks to define a set of skills that may be instantiated in a specific task in a particular domain. As such, this dissertation is part of a growing body of work in the machine-learning and AI literature that makes use of the graphical structure of the data. In the reinforcement-learning literature, a similar graphical approach is taken in defining proto-value functions (Mahadevan 2005, Mahadevan and Maggioni 2007), a set of basis functions for representing value functions. The common starting point of proto-value functions and of access skills is the topology of the state space. The work in this dissertation focuses on how the topology influences navigation in the state space, while the work on proto-value functions focuses on how it constrains the value function. Although the objectives are different, there is overlap in the machinery used. Specifically, the spectral decomposition of the Laplacian of the state transition graph plays an important role both in proto-value functions and in one of the algorithms proposed here.

The graphical representation of the agent's interaction with its environment has been used earlier in the literature by Menache et al. (2002) and by Mannor et al. (2004) to discover useful skills. As noted earlier, these algorithms are among those that are motivated by identifying bottlenecks. They partition the interaction graph to identify regions that are densely connected within themselves, but weakly connected to each other, forming skills that efficiently take the agent to states that border the different regions. Access skills are conceptually different—they are based on an analysis of shortest paths on the interaction graph. How they differ from the skills identified by the algorithms of Menache et al. (2002) and Mannor et al. (2004) is discussed more fully later in the dissertation.

An important dimension in which the work presented in this dissertation differs from a large body of existing work is that it demonstrates the feasibility of forming a large class of useful skills without the need for extensive experience in the domain. Before forming skills, many existing algorithms require the agent to have already learned to perform a number of different tasks in the domain (e.g, Thrun and Schwartz 1995, Pickett and Barto 2002, Stolle and Precup 2002, Stolle 2004, Mehta et al. 2008, Marthi et al. 2007) or to have succeeded at performing a single task many times (e.g., McGovern and Barto 2001, McGovern 2002), or to have knowledge of the task dynamics or to have explored the environment sufficiently to have discovered it approximately (e.g., Menache et al. 2002, Mannor et al. 2004, Mathew 2008, Hengst 2002, Jonsson and Barto 2005). In contrast, here I demonstrate the possibility of forming useful skills without the need to have already performed the task or even to have explored a large part of the environment. This has important implications. Many real-world problems we would like artificial agents to address are so difficult that achieving the desired outcome even once is a challenge beyond the capabilities of today's agents.

CHAPTER 3

ACCESS SKILLS

A necessary first step towards developing effective skill-discovery algorithms is to define what makes a useful skill. For instance, what properties of grasping make it a useful skill for agents that routinely manipulate objects? More importantly, can we express the answer without reference to particular properties of grasping and manipulation tasks, using instead properties that are shared across different tasks and environments?

In this chapter, I propose one answer to the question of what constitutes a useful skill. My answer is based on a graphical representation of an agent's interaction with its environment. Specifically, I use *betweenness*, a measure of centrality on graphs (Freeman 1977, 1979), to define a set of skills that allows efficient navigation on the interaction graph by exploiting its structural properties. In the game of Tic-Tac-Toe, these skills translate into setting up a fork to force the opponent to lose. In the Towers of Hanoi puzzle, they include clearing the stack above the largest disk and clearing one peg entirely to be able to move the largest disk to a different peg.

The primary contribution of this chapter is conceptual. I show that a single concept can capture a wide range of skills in a diverse set of domains. These skills are consistent with common sense, are similar to skills that people handcraft for these domains, and improve learning performance.

A secondary contribution of the present chapter is algorithmic. The skill definition proposed here may be used directly to form a suitable set of skills if the interaction graph is readily available. But, more importantly, this definition is a useful guide

for developing low-cost, incremental skill-discovery algorithms that do not rely on complete or explicit representation of the interaction graph.

The skill characterization presented here captures and generalizes, at least intuitively, the concept of a "bottleneck", which is an early and persistent theme in the skill discovery literature (McGovern and Barto 2001; Stolle and Precup 2002; Stolle 2004; Menache et al. 2002; Mannor et al. 2004; Mathew 2008). The canonical example of a bottleneck is a doorway connecting two rooms. A skill that takes the agent efficiently to a bottleneck is considered a useful behavioral unit, one that allows efficient exploration of possible solutions to many tasks posed in that domain. Bottlenecks have been described in intuitive terms such as *regions that the agent tends to visit frequently on successful trajectories but not on unsuccessful ones* (McGovern and Barto 2001) or *border states of strongly connected areas* (Menache et al. 2002). The explicit and concrete description presented here of what makes a useful skill serves to identify some of the limitations of these existing methods and suggests alternative approaches.

In the following sections, I first present the skill definition I propose, give examples of skills that emerge from this definition in a variety of domains, and empirically test their utility in a number of learning problems. I continue with a discussion of the experimental results, directions for further development of the ideas presented in this chapter, and related work in the literature.[1]

3.1 Access Skills

I argue that states that have a pivotal role in efficiently navigating the interaction graph are useful states to reach and that a useful measure for evaluating how pivotal a vertex v is

[1]The ideas presented in this chapter appear in Şimşek and Barto (2009).

$$\sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}} w_{st}, \tag{3.1}$$

where σ_{st} is the number of shortest paths from vertex s to vertex t, $\sigma_{st}(v)$ is the number of such paths that pass through vertex v, and w_{st} is the weight assigned to paths from vertex s to vertex t.

With uniform path weights, Expression 3.1 defines *betweenness*, a measure of centrality on graphs (Freeman 1977, 1979; Wasserman and Faust 1994). It gives the fraction of shortest paths on the graph, between all possible sources and destinations, that pass through the vertex of interest. If there are multiple shortest paths from a given source to a given destination, they are given equal weights that sum to one.

Path weights are included in Expression 3.1 to take into account the reward function. Depending on the reward function—or a probability distribution over possible reward functions—some parts of the interaction graph may be given more weight than others, depending on how well they serve the agent's needs.

Betweenness may be computed in $O(nm)$ time and $O(n+m)$ space on unweighted graphs with n nodes and m edges (Brandes 2001). On weighted graphs, the space requirement remains the same, but the time requirement increases to $O(nm+n^2 logn)$.

I define *access states* to be those states that correspond to local maxima of betweenness on the interaction graph. They are states that have a higher betweenness than other states in their neighborhood. Here, I use a simple definition of *neighborhood*, including in it only the states that are one hop away in either direction, which may be revised in the future. *Access skills* are behaviors that efficiently reach access states. They may be combined in different ways to efficiently reach different parts of the interaction graph, serving as useful building blocks for efficiently navigating the graph.

The definition of access states contains both a local and a global component. What is important is not the absolute value of betweenness of a given state, but how it compares to other states in its proximity. On the other hand, incorporating path

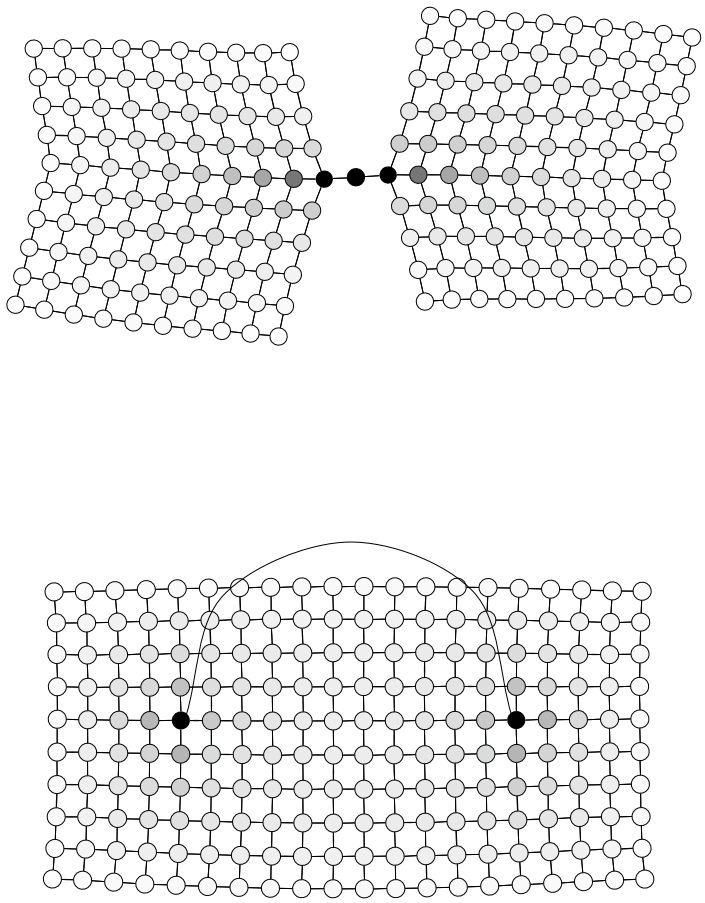


Figure 3.1. A visual representation of betweenness on two sample graphs.

weights in the betweenness computation introduces a global component. While the local component refers to how pivotal a vertex is for connecting its neighborhood to the rest of the graph, the global component refers to whether the regions of the graph to which the vertex provides efficient access are desirable.

Figure 3.1 shows a visual representation of betweenness on two sample graphs, computed using uniform edge and path weights. The gray-scale shading on the vertices corresponds to the relative values of betweenness, with black representing the

highest betweenness on the graph and white representing the lowest. The graph on top corresponds to a gridworld in which a doorway connects two rooms. The graph in the bottom has a doorway of a different type: an edge connecting two otherwise distant nodes on a uniform lattice. In both graphs, states that are local maxima of betweenness correspond to our intuitive choice of subgoals.

3.2 Examples

The skill definition of the previous section was applied to a variety of domains to investigate the type of skills it captures in various different problems. The results are discussed below. Unless stated otherwise, actions had uniform cost, betweenness was computed using uniform path weights, and layouts of interaction graphs were determined by a force-directed algorithm that models the edges as springs and minimizes the total force on the system. To improve visibility, edge directions and self-transitions are omitted from the interaction graphs. The gray shading on the vertices represent the betweenness of the vertex, with black representing the highest betweenness on the graph and white representing the lowest. A vertex was considered to be a local maximum if its betweenness was higher than or equal to those of its immediate neighbors, taking into account both incoming and outgoing edges.

3.2.1 Rooms

The first example is the gridworld domain shown in Figure 3.2. At each state, the available actions are `north`, `south`, `east`, and `west`. These move the agent in the intended direction with probability 0.8 and in a uniform random direction with probability 0.2. If the direction of movement is blocked, the agent remains in the same location. The local maxima of betweenness are the fourteen states that surround the doorways, which have slightly higher betweenness than the doorways themselves.

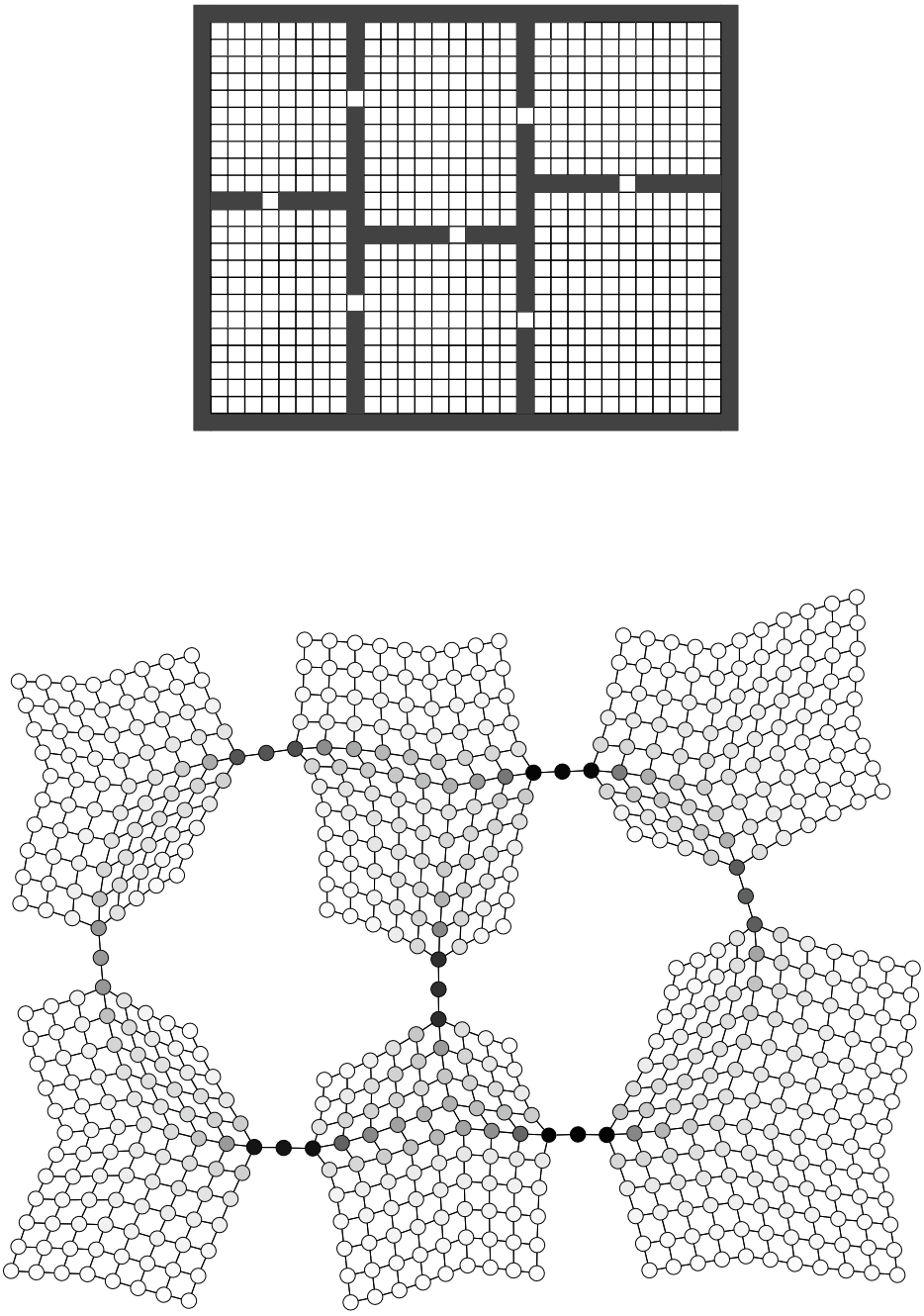


Figure 3.2. The Rooms domain and its interaction graph showing betweenness.

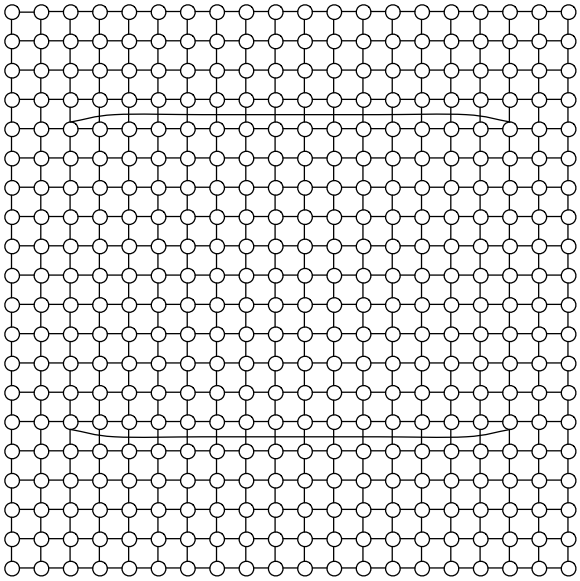


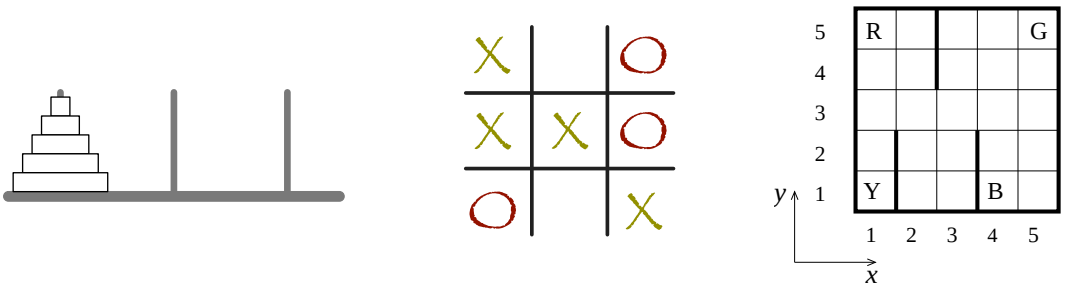
Figure 3.3. The Shortcuts domain.

3.2.2 Shortcuts

The Shortcuts domain consists of two shortcut edges placed on top of a regular lattice as shown in Figure 3.3. The transition dynamics of the domain is similar to that of the Rooms domain, but there is an additional action that can be taken in each of the four states that border the shortcut edges. This action succeeds with probability one in transporting the agent to the other side of the shortcut edge. The local maxima of betweenness in this domain are the four states that border the shortcut edges.

3.2.3 Towers of Hanoi

The Towers of Hanoi puzzle consists of three pegs and a number of disks of different sizes that can slide onto any peg. Each move consists of taking the top disk from one of the pegs and sliding it onto another peg, on top of the other disks that may already be present there. A disk may not be placed on top of a disk that is smaller than itself. Only one disk may be moved at a time. Legal positions are those in which no disk



(a) (b) (c)

Figure 3.4. Three of the domains used in the empirical evaluation. (a) A legal configuration of the Towers of Hanoi puzzle with 5 disks, (b) A final board configuration in the game of Tic-Tac-Toe showing a win for the X player, (c) the Taxi grid.

is placed on top of a disk smaller than itself. Figure 3.4a shows a legal position in a puzzle with five disks.

The corresponding interaction graph is shown in Figure 3.5. The highest local maxima of betweenness divide the graph into three clusters in which the largest disk is always on the same peg. Skills that take the agent to these maxima correspond to setting up the pegs so that the largest disk may be moved to a different peg and, in some cases (depending on the direction of movement on the graph), also executing this move. The setup for this move involves clearing the stack on top of the largest disk (so that it is legal to move the largest disk) and clearing another peg entirely (so that it is legal to move the largest disk to that peg).

Within each of the three clusters, there are additional local maxima of betweenness. The next highest local maxima correspond to an analogous movement with the second largest disk. Figure 3.6 shows all local maxima in the domain, omitting symmetries with respect to peg identities. The states are ordered from left to right in decreasing betweenness.

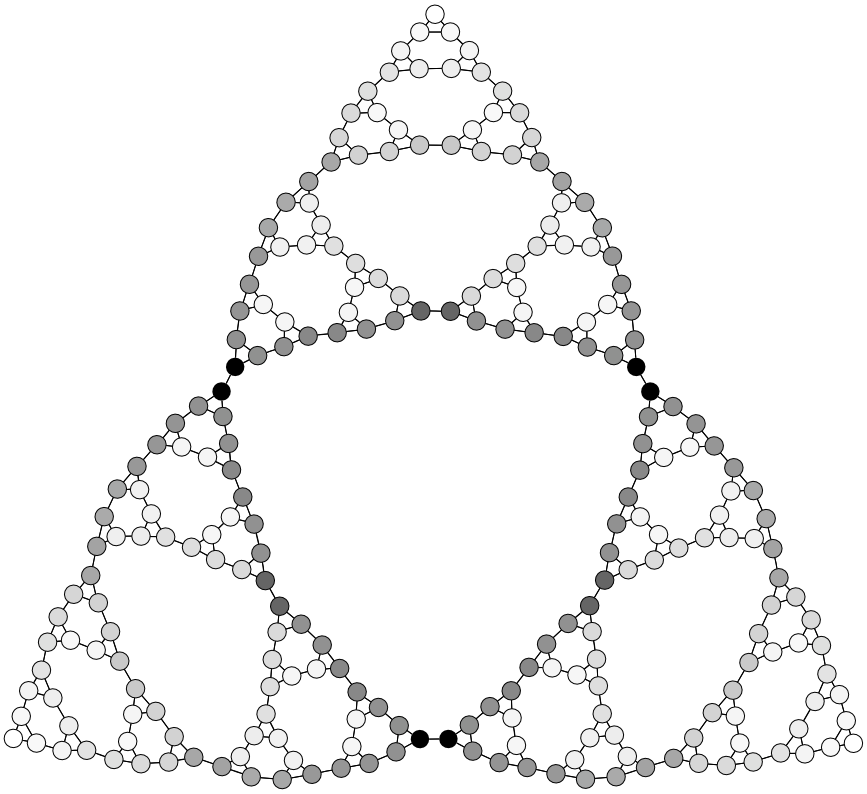
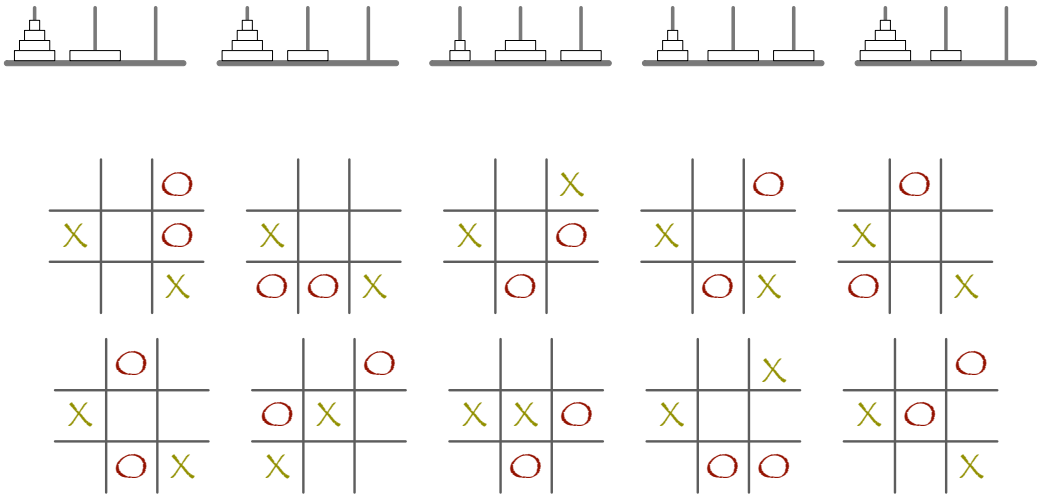


Figure 3.5. Betweenness in the Towers of Hanoi puzzle with 5 disks.



Eye	Hand	Marker	Light	Music	Monkey	Bell
ball	ball	bell	off	on	quiet	off
light switch	light switch	bell	on	on	quiet	off
music button	music button	bell	on	off	quiet	off
light switch	light switch	bell	off	on	frightened	off
light switch	light switch	bell	on	off	quiet	off
bell	bell	bell	off	on	frightened	off

Taxi state (x y Passenger* Destination)							
02TY	30TB	04TR	44TG	12TR	22TG	02YB	02YG
02YR	04TB	04TY	44TB	04TG	44TR	44TY	12RB
12RG	12RY	32BG	32BR	32BY	22GB	22GR	22GY
*T denotes "in-taxi".							

Figure 3.6. Local maxima of betweenness in the Towers of Hanoi puzzle, the game of Tic-Tac-Toe, the Playroom domain, and the Taxi domain.

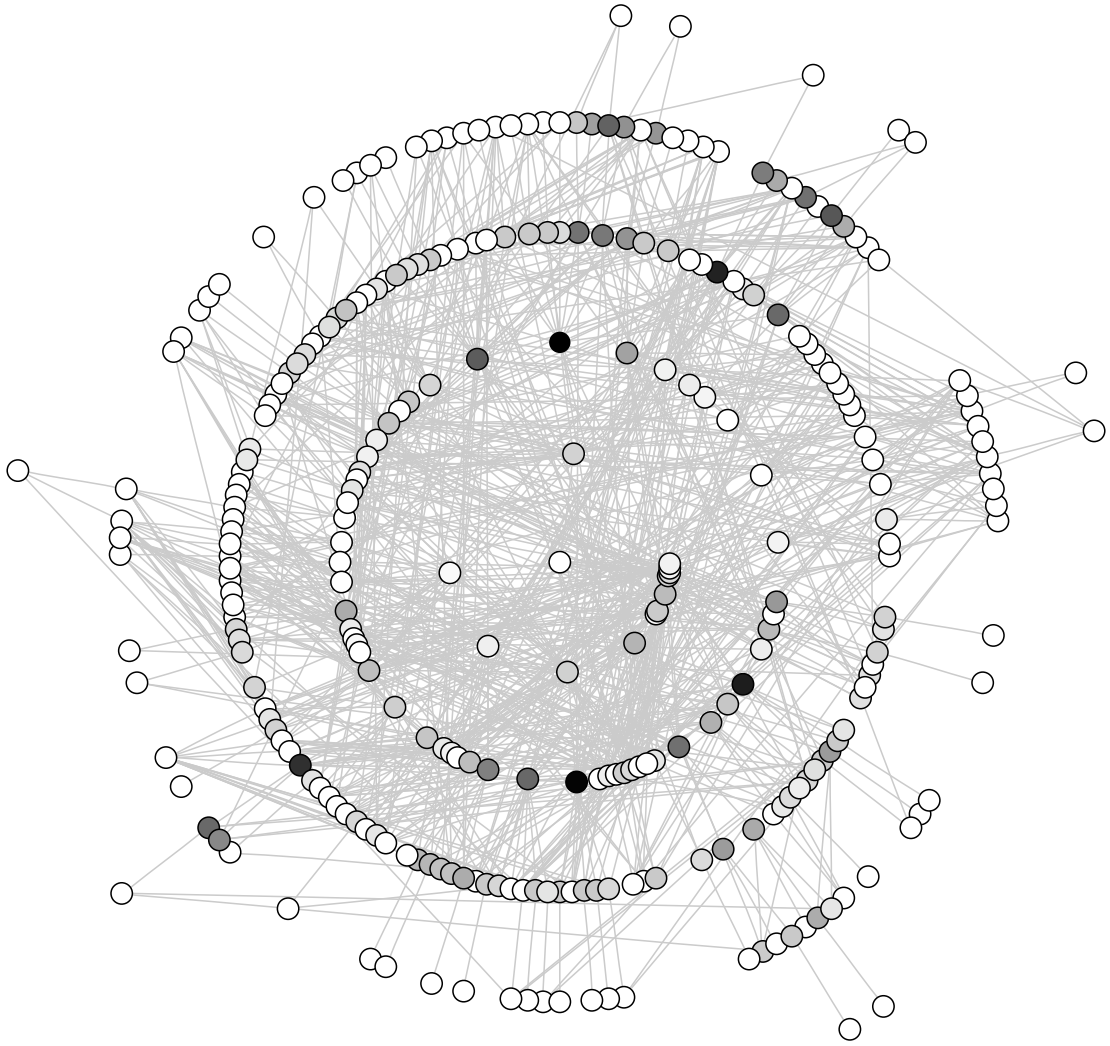


Figure 3.7. Betweenness in the game of Tic-Tac-Toe.

3.2.4 Tic-Tac-Toe

Tic-tac-toe is a board game in which two players take turns to placing marks on a 3×3 board. The player who succeeds in marking a horizontal, vertical, or diagonal row wins the game. Figure 3.4b shows a final board configuration showing a win for the X player. In the current implementation, the agent played first. The opponent followed a policy that (1) placed the third mark in a row, whenever possible, winning the game, (2) blocked the agent from completing a row, or (3) placed its mark on a random empty square, with decreasing priority. The state representation was invariant with respect to rotational and reflective symmetries of the board.

Figure 3.7 shows the interaction graph. The node at the center is the empty board position. Other states form rings around the center node with respect to their path length from this initial configuration, with the innermost ring showing states in which both players have had a single turn.

To compute betweenness, a weight of +1 was assigned to paths that terminate at a win for the agent and 0 to all other paths. Figure 3.6 shows the local maxima of betweenness.[2] The agent is the X player and will go next. Each of these states gives the agent the possibility of immediately setting up a fork, creating an opportunity to win in the next turn.

3.2.5 Playroom

The Playroom domain (Barto et al. 2004; Singh et al. 2005) contains an agent that interacts with a number of objects in its surroundings. A Markov version of the domain was used here which contains a light switch, a ball, a bell, a button for turning music on and off, and a toy monkey. The agent has an eye, a hand, and a marker it can place on objects. Its actions include looking at a randomly selected

[2]There were five other "trivial" local maxima, which were states that allow the agent to immediately win the game by placing the third X in a row.

object, looking at the object in its hand, holding the object it is looking at, looking at the object that the marker is placed on, placing the marker on the object it is looking at, moving the object in its hand to the location it is looking at, flipping the light switch, pressing the music button, and hitting the ball towards the marker. The first two actions succeed with probability 1, while the remaining actions succeed with probability 0.75, producing no change in the environment if they fail. In order to operate on an object, the agent must be looking at the object and holding the object in its hand. To be able to press the music button successfully, the light should be on. The toy monkey starts to make frightened sounds if the bell is rung while the music is playing; it stops only when the music is turned off. If the ball hits the bell, the bell rings for one decision stage.

The MDP state consists of the object that the agent is looking at, the object that the agent is holding, the object that the marker is placed on, music (on/off), light (on/off), monkey (frightened/quiet), and bell (ringing/off).

The interaction graph of this domain is shown in Figure 3.8. To the best of my knowledge, the Playroom domain has not been used for skill discovery in the literature and its graphical structure has not been examined previously.

The six different clusters of the interaction graph emerge naturally from the force-directed layout algorithm and correspond to the different settings of the music, light, and monkey variables. There are only six such clusters because not all combinations are possible. The graph shows that betweenness values peak at regions that immediately connect neighboring clusters, corresponding to skills that change the settings of the music, light, or monkey variables. It is a surprising and pleasing finding that

the graphical structure of the domain naturally corresponds to skills that have been user-specified in the earlier work of Barto et al. (2004) and Singh et al. (2005)[3].

3.2.6 Taxi

The taxi domain (Dietterich 2000) includes a taxi and a passenger on a 5×5 grid shown in Figure 3.4c. At each grid location, the taxi has six primitive actions: `north`, `east`, `south`, `west`, `pick-up`, and `put-down`. The navigation actions succeed in moving the taxi in the intended direction with probability 0.80; with probability 0.20, the action takes the taxi to the right or left of the intended direction. If the direction of movement is blocked, the taxi remains in the same location. The action `pick-up` places the passenger in the taxi if the taxi is at the passenger location; otherwise it has no effect. Similarly, `put-down` delivers the passenger if the passenger is inside the taxi and the taxi is at the destination; otherwise it has no effect. The source and destination of all passengers are chosen uniformly at random from among the grid squares R, G, B, Y. Here, I used a continuing version of this domain in which a new passenger appears after each successful delivery.

Figure 3.9 shows the interaction graph. The highest local maxima of betweenness are at four regions of the graph that correspond to passenger delivery. Other local maxima belong to one of the following categories: (1) taxi is at the passenger location[4], (2) taxi is at one of the passenger wait locations with the passenger in the taxi[5], (3) taxi and passenger are both at destination, (4) the taxi is at location $x = 2, y = 3$, a navigational bottleneck on the grid, and (5) the taxi is at location $x = 3, y = 3$,

[3]In this earlier work, the agent creates skills that change the settings of the salient variables in the environment. Because salient variables are defined by the system designer, the algorithm in this earlier work does not *discover* skills in the sense of the term used here.

[4]Except when passenger is waiting at Y, in which case the taxi is at grid location $x = 1, y = 3$.

[5]For wait location Y, the corresponding subgoal is taxi is at grid location $x = 1, y = 3$, having picked up the passenger.

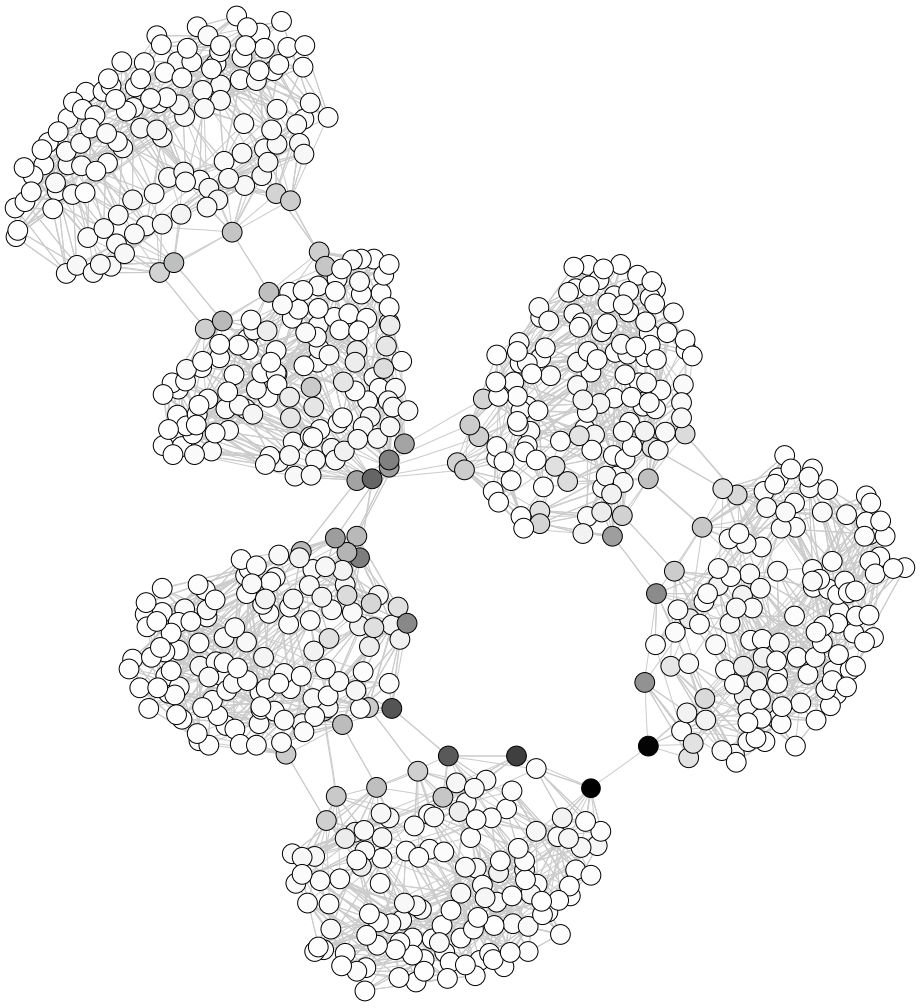


Figure 3.8. Betweenness in the Playroom domain.

another navigational bottleneck. The corresponding skills are (approximately) those that take the taxi to the passenger location, to the destination with passenger in the taxi, or to a navigational bottleneck. These skills closely resemble those that are hand-coded for this domain in the literature, for example, in Dietterich (2000), but without the state abstraction. All local maxima are shown in Figure 3.6.

3.3 Empirical Evaluation

The previous section showed that the skill definition of Section 3.1 translated into skills that are intuitively appealing in a variety of domains. Here, I carry out an empirical investigation to evaluate how these skills impact learning performance when made available to a reinforcement-learning agent.

In all experiments reported, the agent used Q-learning with ϵ-greedy exploration, with ϵ set to 0.05. When using skills, it performed both intra-option and macro-Q updates. The learning rate (α) was kept constant at 0.1. Initial Q-values were 0. Discount rate γ was set to 1 in episodic tasks, to 0.99 in continuing tasks.

Skills were made available to the agent fully formed at the beginning of the learning trials. Unless stated otherwise, a single skill was created for each access state in a domain, eliminating redundancies by selecting a single representative among access states that are in close proximity (within 2 hops). The exact set of skills that were tested in each domain is shown in Table 3.1. The initiation sets of the skills were restricted to include a certain number of states. This number varied between experiments, but in any single experiment, all skills had initiation sets of the same size. The initiation sets consisted of the specified number of states that had the smallest hop distance to the skill destination on the interaction graph, with ties broken randomly. The skills terminated with probability one outside their initiation set and at their destination; they continued to execute in all other states. The skill policy was the optimal policy for reaching the skill destination.

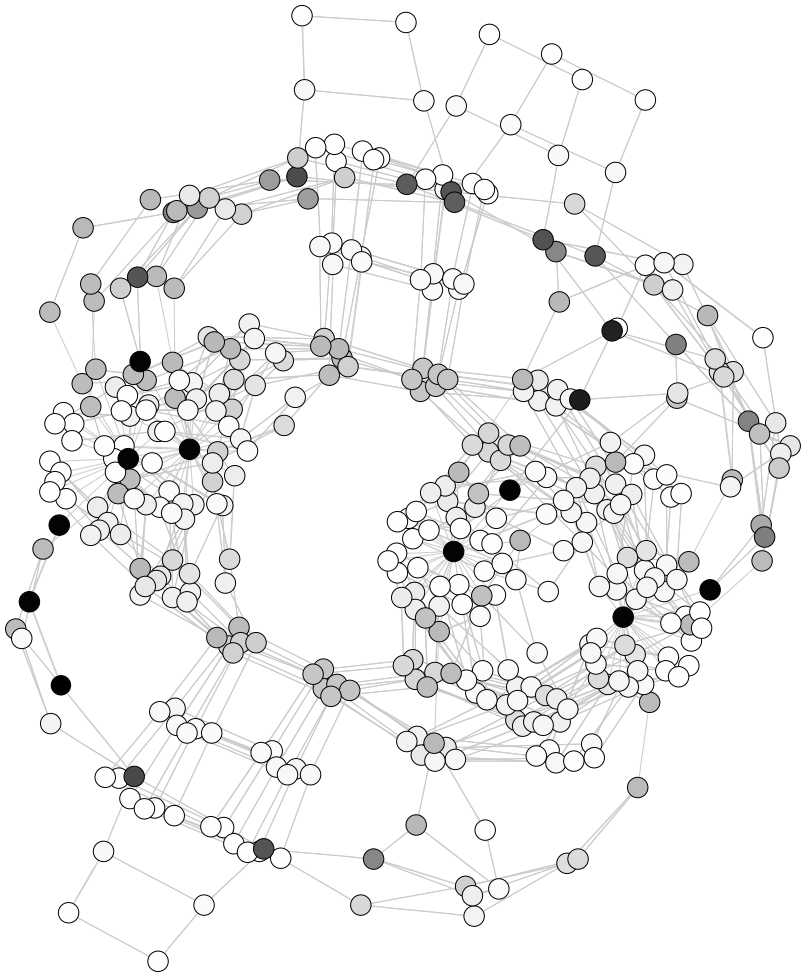


Figure 3.9. Betweenness in the Taxi domain.

In Rooms, Shortcuts, Towers of Hanoi, and Playroom domains, access skills were tested on a collection of 500 tasks. Each task had a single, randomly-selected goal state. The initial states were selected randomly. Domain reward consisted of -0.001 for each transition. Task reward was $+1$ for transitions into the the goal state, 0 for all other transitions. In Towers of Hanoi, to make the learning problem more difficult, actions were implemented with stochastic effects. An action succeeded as intended with probability 0.8; otherwise, it resulted in a random slip of the disk into any of the legal pegs. The interaction graph and its access states of this stochastic version are identical to those of the deterministic game.

Figure 3.10 shows learning performance, comparing an agent using only primitive actions to an agent using both primitive actions and skills. The figure shows mean performance in the 500 tasks that were tested. In all four domains, the availability of access skills resulted in a dramatic improvement in performance. The numbers in the legends show the mean number of skills available in a state. For example, in the Rooms domain, 1 skill/state indicates that the initiation sets consisted of 89 states, 2 skills/state indicates that the initiation sets consisted of 178 states, and so on.

Figure 3.11 shows additional results in these domains. These graphs include comparisons with an agent that used a control group of skills whose destinations were selected randomly. In all performance results that compare access skills to random

Table 3.1. Domains and skills used in the empirical evaluation.

Domain	Vertices	Edges	Skills	Skill destinations
Rooms	623	2461	7	Doorways
Shortcuts	400	1600	4	States that border the shortcut edges
Playroom	755	5055	6	States in Figure 3.6
Towers of Hanoi	243	726	21	States in Figure 3.6 and states that are symmetric to them with respect to peg identities but are not adjacent to them
Tic Tac Toe	296	965	1	States in Figure 3.6
Taxi	404	1636	24	States in Figure 3.6

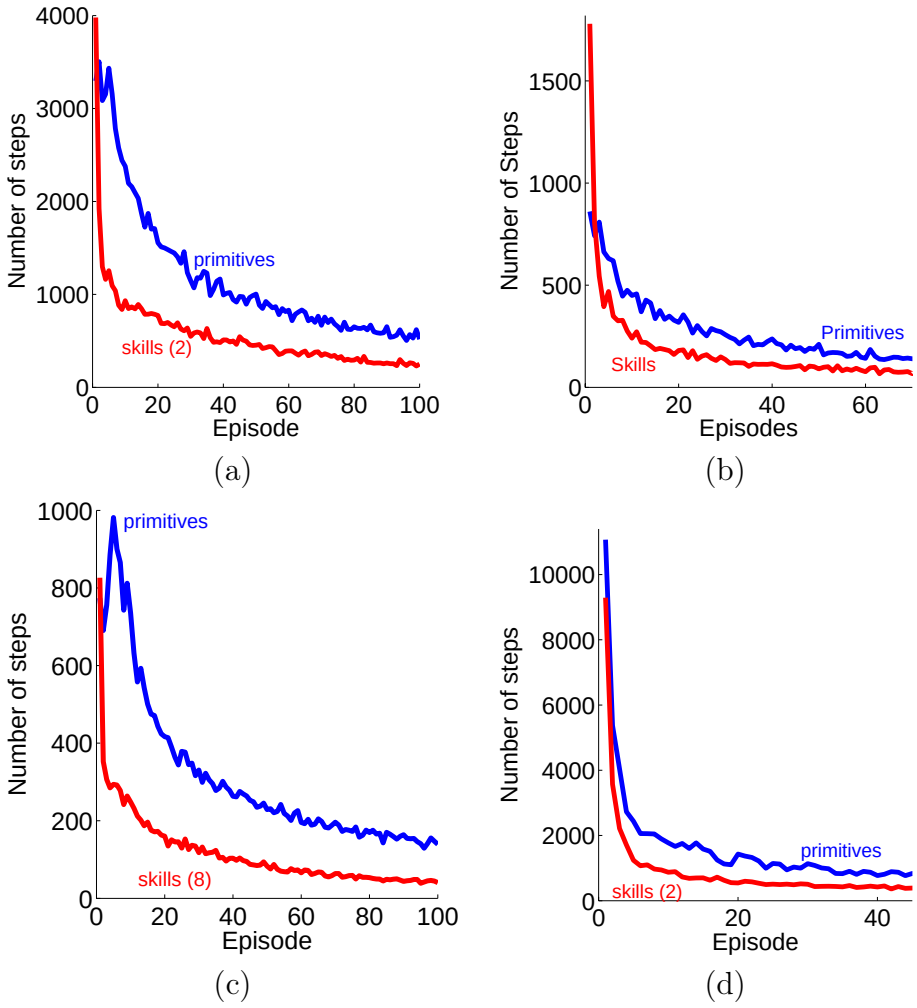


Figure 3.10. Performance with and without access skills in the (a) Rooms, (b) Shortcuts, (c) Tower of Hanoi, and (d) Playroom domains.

skills, the number of skills used and the size of the initiation sets were identical. The plots show cumulative number of steps as a function of episodes completed, for easier comparison of the different learning curves.

These plots show that access skills improved Q-learning performance compared to both the primitives-only setting and the random-skill setting, for different settings of the initiation set size. Random skills usually improved performance compared to the primitives-only setting, but this improvement was much smaller than that obtained by access skills. In general, performance improved with increases in initiation set size.

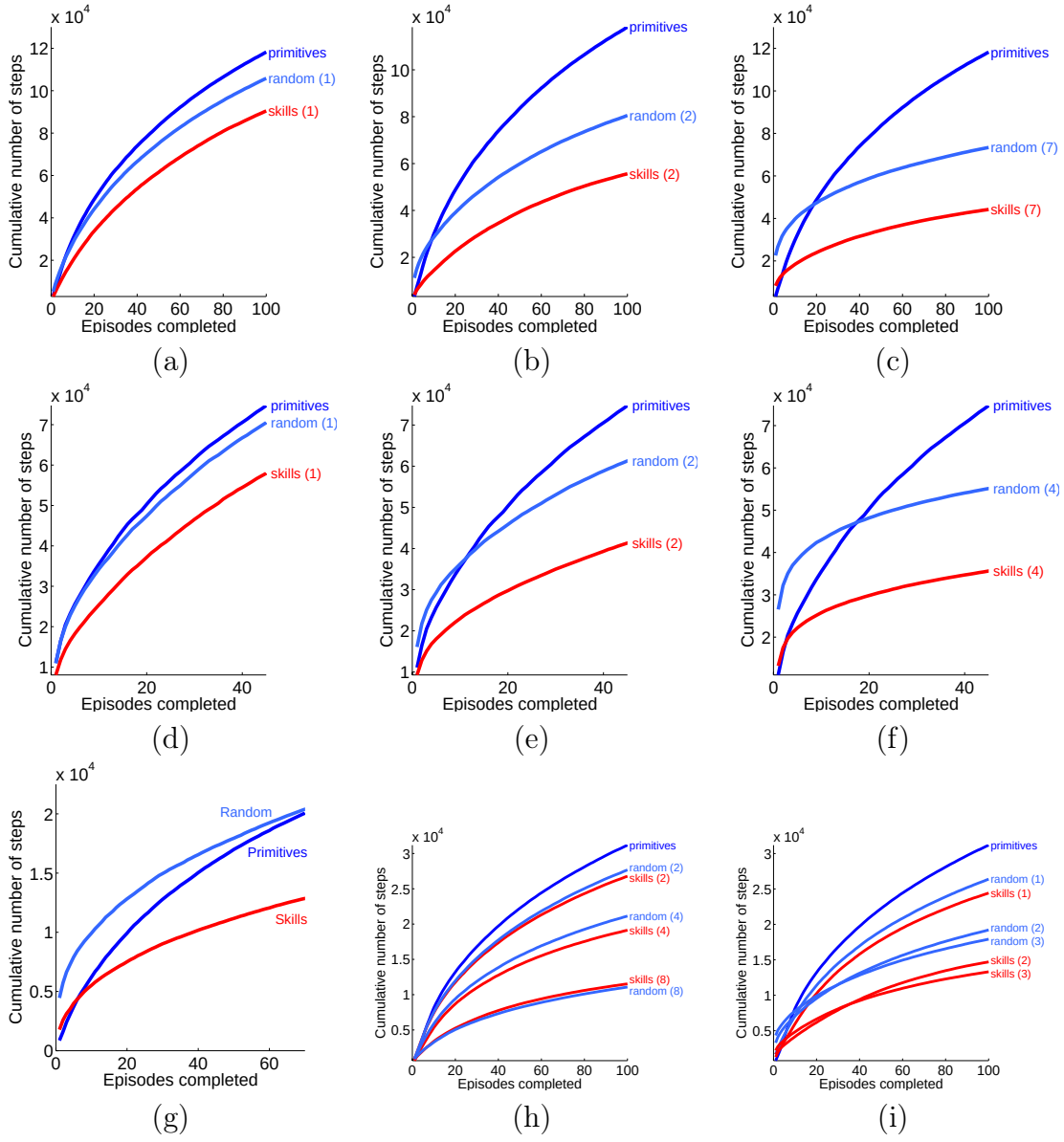


Figure 3.11. Additional performance results. (a) (b) (c) Rooms, (d) (e) (f) Playroom, (g) Shortcuts, (h) Towers of Hanoi, (i) Towers of Hanoi with only the three skills that correspond to global maxima of betweenness.

In the Taxi domain, performance was evaluated in a single continuing task that rewarded the agent for continuously picking up and delivering passengers. Reward was -1 for each action, an additional $+50$ for passenger delivery, and an additional -10 for an unsuccessful `pick-up` or `put-down` action. Figure 3.12 shows learning performance in 500 trials. The results are qualitatively similar to those obtained in earlier domains.

In Tic-Tac-Toe, performance was similarly evaluated on a single task. Reward was -0.001 for each action, an additional $+1$ for winning the game, and an additional -1 for losing. Creating an individual skill for reaching each of the access states (as done in other domains) generates skills that are not of much use in Tic-Tac-Toe. It is almost always impossible to reach a desired board configuration. A skill with a single destination would typically need to be abandoned after only one action. Therefore, a single skill with multiple destinations was formed in this domain. The destinations were the local maxima of betweenness shown in Figure 3.6. The initial Q-value of this skill was set to 1 at the start state to ensure that it got executed frequently enough. It was not clear what would be a meaningful control condition for this single skill. Therefore, there is no control condition with random skills in this domain. Figure 3.13 shows mean performance in 100 trials, revealing a large improvement in performance when the skill was available.

3.4 Related Work

A graphical approach to forming skills was first suggested by Amarel (1968) in his classic analysis of the missionaries and cannibals problem. Amarel advocated representing action consequences in the environment as a graph and forming skills that correspond to navigating this graph by exploiting its structural regularities. He did not, however, propose any general mechanism that can be used for this purpose.

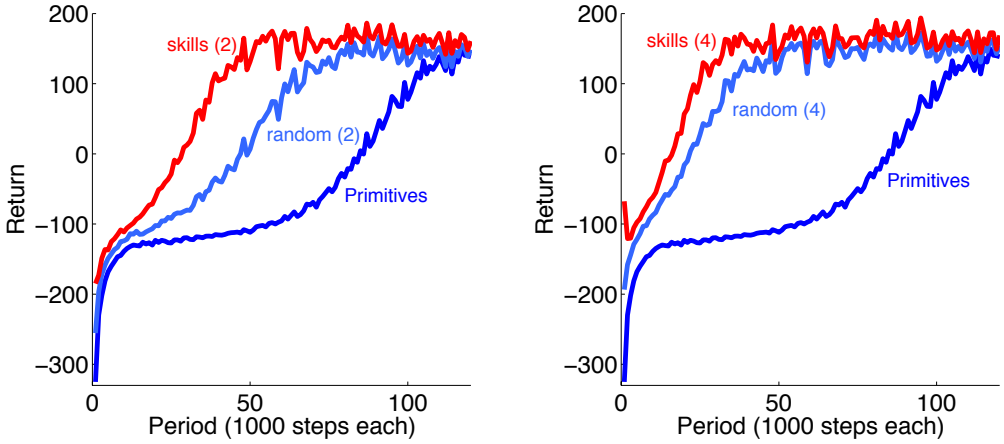


Figure 3.12. Performance in the Taxi domain.

The skill definition proposed here captures the "bottleneck" concept, which has inspired many of the existing skill-discovery algorithms, most prominently the algorithms by McGovern and Barto (2001), Stolle and Precup (2002), Stolle (2004), Menache et al. (2002), Mannor et al. (2004), and Mathew (2008). There is clearly an overlap between the skills proposed here and the skills that are formed by these discovery algorithms. For example, in gridworld environments, all of these discovery algorithms generate skills for efficiently reaching doorways, which also emerge from the skill definition provided here. Below, I review each of these algorithms in turn, focusing on whether they can identify access states and, if so, whether they do it efficiently.

The discovery method of McGovern and Barto (2001) examines past trajectories to identify states that are common in successful trajectories but not on unsuccessful ones. The definition of "success" is left to the system designer. In their implementation, successful trajectories are defined to be those in which the agent reaches a goal state within a certain number of decision stages.

A fundamental property of this algorithm prevents it from identifying a large subset of access states: It examines different paths that reach the same goal location,

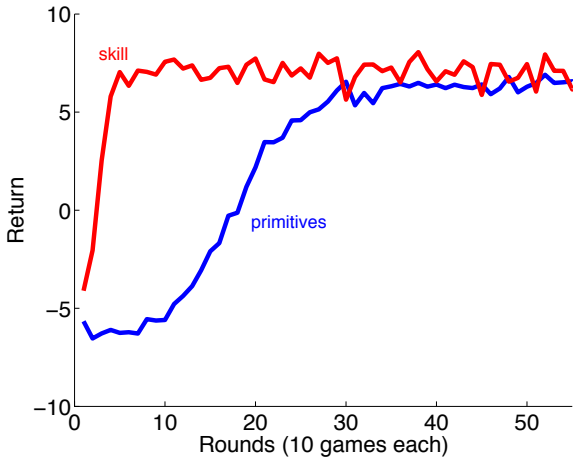


Figure 3.13. Performance in the game of Tic-Tac-Toe.

while the definition of access states consider the most efficient ways of navigating between different source and destination pairs. Access states that are not on the path to the goal state would not be identified by the method of McGovern and Barto (2001).

With respect to sample complexity, an important concern with the approach is its need for excessive exploration of the environment. Their method can be applied only after the agent has successfully performed the task at least once. Typically, it requires many additional successful trajectories to be able to reliably distinguish bottlenecks from other states. There is reason to believe that the information they ignore—how the states in a trajectory are linked together—may be the basis of a much faster and accurate algorithm. Even a short trajectory may reveal enough information about the graphical structure of the environment to suggest useful subgoals.

As a preliminary test of this conjecture, I replicated the experiment on the two-room gridworld in McGovern and Barto (2001), using the trajectories differently. The state trajectory from the first episode was used to construct the interaction graph to compute betweenness. In 774 of 1000 trials, the state with the highest betweenness was either a doorway state or a state within one transition away from the doorway. At

the end of the second episode, this number increased to 954. In contrast, in McGovern and Barto (2001), the subgoal discovery process starts after 25 episodes.

To summarize, the discovery method of McGovern and Barto (2001) can identify only a small subset of access states and that the discovery process has a higher sample complexity than may be possible using alternative approaches.

Stolle and Precup (2002) and Stolle (2004) build on the work by McGovern and Barto (2001) to identify a broader set of bottleneck states. They propose a number of algorithms that mine past trajectories using visitation frequencies, obtaining their trajectories from multiple tasks that start and terminate at different states. As the number of tasks increases, the subgoals that are identified become more and more similar to access states. Unfortunately, however, data efficiency is even a larger concern with their algorithms. They require the agent to have already identified the optimal policy—not for only a single task, but for many different tasks in the domain.

Menache et al. (2002) and Mannor et al. (2004) have proposed discovery techniques that address the high sample complexity of frequency-based approaches. These algorithms make use of the graphical structure of the problem, but their use of graphical structure differs from the approach taken here. These algorithms construct the interaction graph from experience, apply a clustering algorithm to partition the graph into blocks and create skills that take the agent efficiently to states that connect different blocks. The objective is to identify blocks that are highly connected within themselves but weakly connected to each other. Different clustering techniques and cut metrics may be used towards that end.

The Shortcuts domain illustrates the limitations of these techniques in identifying access states. There are access states in this domain, but unlike a doorway, they do not partition the state space into two weakly-connected regions. The cut that minimizes the widely-used *NCut* and RatioCut metrics (defined in Section 4.3.2) cuts the graph horizontally in the middle as shown in Figure 3.14.

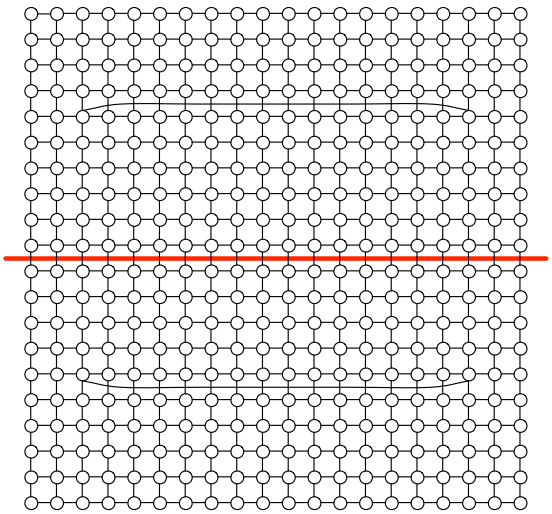


Figure 3.14. A cut of the Shortcuts domain that minimizes both *NCut* and *RatioCut* metrics.

Although it is motivated from a different perspective, the algorithm by Mathew (2008) also partitions the interaction graph and therefore shares the limitations of the algorithms by Menache et al. (2002) and Mannor et al. (2004).

The argument against graph partitioning extends beyond the simple example provided by the Shortcuts domain. Most real-world domains show a complex connectivity structure that does not lend itself to the use of graph partitioning in identifying the access states. For example, analogous situations exist in continuous control problems where sequential composition of "funnels" in system dynamics can give rise to access-like states (Burridge et al. 1999).

The more fundamental property that makes a doorway a useful subgoal is that it is *between* many source-destination pairs. Discovery methods that partition the interaction graph can not directly tap into this property, although they can sometimes do it indirectly. In the following chapter, I propose a discovery algorithm based on graph partitioning that is better suited for identifying access states than the existing partitioning approaches.

3.5 Discussion

In a diverse set of domains, the skill definition of Section 3.1 gives rise to skills that make common sense and that are similar to skills people handcraft for these domains. When added to the action repertoire of an agent, these skills improved learning performance, to a greater extent than a control group of randomly generated skills, suggesting that the improvements should not be attributed to the presence of skills alone but that the specific skills that were formed are valuable.

In most cases, the experimental evaluation showed an improvement in performance when using skills with randomly-selected destinations compared to the primitives-only condition. This result warrants further study of random selection of skill destinations. Although judicious selection results in better performance, the low computational cost of random selection may make it attractive in some circumstances.

The skill definition proposed here can take into account the reward function—or a probability distribution over possible reward functions—through assigning different weights to different paths on the graph. This property is desirable because the utility of a skill depends critically on the reward function. Two identical agents operating in the same environment but performing different tasks require different sets of skills suitable for the individual tasks that they face. Instead of an external reward function, path weights may reflect an agent's internal state, more specifically, an agent's own learning priorities in the current stage of its development.

Although knowledge about the reward function may be taken into account, when such information is available, it is not necessary for the approach to specify useful skills. The experimental evaluation in this chapter spanned a range with respect to the role played by the reward function in defining the skills. In Tic-Tac-Toe, the reward function was fully taken into account, producing skills that were tailored to the task. In Taxi, the reward function was ignored entirely—betweenness was computed using uniform edge and path weights. The resulting skills were then tested on the

standard reward function for this domain in the literature. In the remaining domains, the action costs were taken into account (or they were identical) but not the location and magnitude of the goal reward. The results demonstrate that the resulting skills were useful in a collection of tasks in which the goal varied drastically, although the skills were not tailored to any of the particular goal locations.

The skill definition proposed here may be refined in a number of ways. First, while the analysis of shortest paths to the exclusion of all others has proved useful, considering a broader set of paths may improve skill utility. Second, action effects may be more accurately represented on the interaction graph, for instance, by including vertices to represent individual actions. Third, the size and constituents of the initiation sets may be informed by the betweenness of the skill destination. For instance, the size of the initiation set may be informed by the absolute betweenness of the skill destination, with larger initiation sets attached to skills that reach globally more important states.

The skill description I propose may be used directly to form a set of skills for a given MDP. However, it may require more information than is available to the agent. Therefore, there is a need to develop methods that rely on less information, in particular, methods that do not require that the interaction graph is known in its entirety. A key property of the approach taken here is that the utility of a state as a subgoal is determined in relation to other states in its vicinity. What matters is not the absolute value of betweenness but whether it is higher than the betweenness values in its proximity. Although betweenness of a vertex can not be estimated reliably using only local information, it should be possible to reliably determine local maxima of betweenness without knowledge of the full graph. I investigate this possibility in the next chapter.

The definition provided here can not directly handle multi-dimensional, continuous states spaces, but it may be extended in the future for such domains. My conjecture

is that the fundamental reason why grasping is a useful skill is analogous: it is *between* various things that we do in our daily life.

It is unlikely that a single skill definition would capture all the different types of skills that an autonomous agent may find useful. I propose the skill characterization of the current chapter as one of several different types of skills that would complement each other.

CHAPTER 4

IDENTIFYING ACCESS SKILLS

The skill definition provided in the previous chapter may be used directly to form a set of skills suitable for a given task or a domain. However, because of its reliance on complete knowledge of the interaction graph and the computational cost of betweenness, the use of the skill definition itself as a discovery method is limited, although there are conditions under which it would be useful.

In this chapter, I develop three alternative approaches for identifying access skills that do not require explicit or complete representation of the interaction graph. All three methods have low computational cost that is not directly related to the number of states in the domain. A preliminary analysis suggests that they are effective in reliably identifying access states.

I assume that the agent can sample paths from the graph through interacting with its environment: executing an action, observing the change in environment state, and receiving a numerical reward signal. The methods I propose are founded on the following observations:

1. On interaction graphs constructed from short trajectories, access states are more likely than other states to be local maxima of betweenness,

2. On interaction graphs constructed from short trajectories, access states are more likely than other states to lie between two regions that are densely connected within themselves but weakly connected to each other,

3. Access states are more likely than other states to introduce short-term novelty, in other words, to mediate a transition from a region recently well explored to a region recently unexplored.

In the following sections, I first discuss the use of the skill definition itself as a discovery method. I then describe the three alternative approaches, discuss their strengths and limitations, and conclude with the contributions of the present chapter.[1]

4.1 Betweenness

The skill definition of Section 3.1 may be used directly to form a set of skills suitable for a given task if the interaction graph of the domain is known and the cost of computing betweenness is not a burden. Knowledge of the reward function is not necessary for this approach to yield useful skills, but the reward function may be fruitfully taken into account to form skills that are tailored to the task at hand. One setting in which this approach would be beneficial is when the agent faces a sequence of different tasks in the same domain, where the computational cost of forming skills would be worthwhile because of the benefits it would introduce over time in different tasks.

4.2 Local Betweenness

The betweenness value of a vertex is a global graph property that can not be estimated reliably without knowledge of the entire graph. It should, however, be possible to reliably determine local maxima of betweenness using only partial information. Of particular interest here is whether it is possible to reliably determine local maxima of betweenness using subgraphs of the interaction graph. Specifically, we are interested

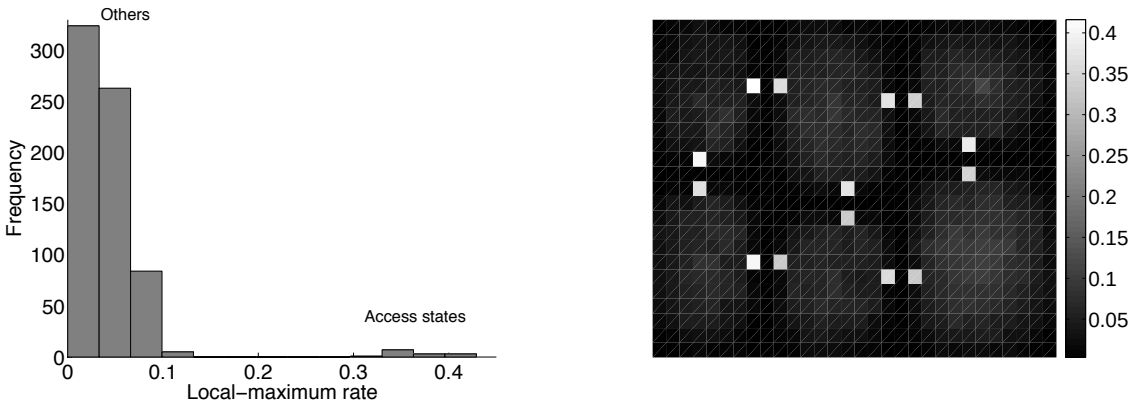
[1]Some of the ideas presented in this chapter have appeared in Şimşek and Barto (2004) and Şimşek, Wolfe and Barto (2005).

in subgraphs that correspond to sample trajectories of the MDP, which an agent interacting with its environment can easily obtain. Other types of subgraphs, for instance, a subgraph around a particular node, containing all nodes within a specified distance from this node, may better support our objective, but may not be easy to obtain for an agent.

> **Definition** A *local interaction graph* is a weighted, directed graph constructed from an input trajectory of states, actions, and rewards sampled from an MDP. Vertices in the graph correspond to states in the trajectory. Edges correspond to state transitions that take place in the trajectory. A directed edge $v_1 \rightarrow v_2$ is present in the graph if and only if the corresponding state transition is present in the trajectory. Edges have two sets of weights defined on them. The first weight is the mean reward obtained in the trajectory when the corresponding state transition took place. The second weight is the number of corresponding transitions that took place in the trajectory.

I hypothesize that the states that are local maxima of betweenness on the full interaction graph are more likely than other states to be local maxima of betweenness on local interaction graphs. Local interaction graphs obtained in the Rooms domain of Section 3.2.1 provide some empirical support for this hypothesis. The agent performed 10,000 random walks of 1000 steps in this domain, each starting at a random location. At the end of each random walk, a local interaction graph was constructed that reflected the agent's brief experience in the domain. The mean number of vertices in these graphs was 159, much smaller than the number of states in the domain.

Figure 4.1a shows the histogram of local-maximum rates obtained, where *local-maximum rate* of a state is the number of local interaction graphs in which the state was a local maximum of betweenness divided by the number of local interaction graphs on which the state was represented. As in the previous chapter, a state was considered to be a local maximum if its betweenness was greater than or equal to those



(a) (b)

Figure 4.1. Local-maximum rates in the Rooms domain.

of its immediate neighbors. The figure shows a bimodal distribution. The majority of states have rates that are below 0.12, while a small number of states have rates that are higher than 0.32. The latter group consists of the fourteen states that are adjacent to doorways, which are the local maxima of betweenness on the full interaction graph. The local-maximum rate of each state is depicted visually in Figure 4.1b, where the gray shading on each grid square represents the local-maximum rate of that state.

The figures show that most states appeared as a local maximum on at least some local interaction graphs. Consequently, for the objective of determining local maxima of the full interaction graph, any given local interaction graph is not particularly informative. In fact, it could be quite misleading. It is possible, however, to combine the evidence from many such graphs to reliably determine the access states of the full graph, as described in the next section.

4.2.1 Formulation As a Classification Problem

The discovery problem may be formulated as a classification task in which the agent's objective is to determine whether a given state is an access state, based on evidence from a collection of local interaction graphs obtained in the domain.

Each local interaction graph yields an observation for each state represented on it: a positive observation if the state is a local maximum of betweenness on the graph, a negative observation otherwise. Both positive and negative observations may be associated with each state, but access states have a higher probability than other states of producing positive observations.

If we make the simplifying assumption that the environment contains two classes of states, *targets* (access states) and *non-targets*, and that states in the same class have the same probability of producing a positive observation, the problem faced by the agent is to classify a given state as target (T) or non-target (N) based on a number of binary observations on that state.

If class-conditional distributions are known, this classification task is straightforward using Bayesian decision theory (Duda et al. 2001). Assigning an appropriate cost to two possible types of error—classifying a target as non-target (miss) or a non-target as target (false alarm)—and minimizing total cost gives rise to a decision rule that labels the state as target if

$$\frac{P(o_1..o_n|T)}{P(o_1..o_n|N)} > \frac{\lambda_{fa}}{\lambda_{miss}} \frac{P(N)}{P(T)},$$

where $o_1..o_n$ are the observations on a given state, $P(T)$ is the prior probability of a target state, $P(N)$ is the prior probability of a non-target state, λ_{fa} is the cost of a false alarm, and λ_{miss} is the cost of a miss. If we assume that the observations are independent and identically distributed given the class, we can rewrite this inequality as

$$\frac{p^{n_+}(1-p)^{n-n_+}}{q^{n_+}(1-q)^{n-n_+}} > \frac{\lambda_{fa}}{\lambda_{miss}} \frac{P(N)}{P(T)},$$

where n_+ is the number of positive observations, n is the total number of observations, p is the probability of a positive observation given a target, and q is the probability

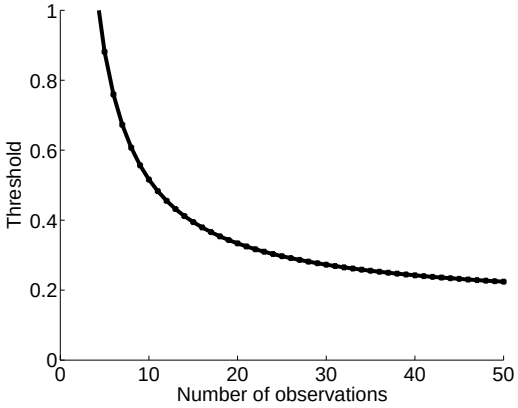


Figure 4.2. Decision threshold as specified by Inequality 4.1 when $\frac{\lambda_{fa}}{\lambda_{miss}} = 100$, $\frac{p(N)}{p(T)} = 100$, $p = 0.36$, and $q = 0.036$.

of a positive observation given a non-target. Simple algebra yields a decision rule to label the state as a target if

$$\frac{n_+}{n} > \frac{\ln\frac{1-q}{1-p}}{\ln\frac{p(1-q)}{q(1-p)}} + \frac{1}{n}\frac{\ln(\frac{\lambda_{fa}}{\lambda_{miss}}\frac{p(N)}{p(T)})}{\ln\frac{p(1-q)}{q(1-p)}}. \tag{4.1}$$

Inequality 4.1 is a threshold on the proportion of positive observations on a given state. The threshold depends on the number of observations collected on the state. With increasing number of observations, the threshold decreases at a decreasing rate, approaching a constant in the limit. Figure 4.16 shows the threshold when $\frac{\lambda_{fa}}{\lambda_{miss}} = 100$, $\frac{p(N)}{p(T)} = 100$, and p and q are estimated using the random walk data in the Rooms domain discussed in Section 4.2 ($p = 0.36$, $q = 0.036$). The figure does not show threshold values greater than 1, the largest value $\frac{n_+}{n}$ can take.

Inequality 4.1 can be used directly as a decision rule if p, q, and $\frac{\lambda_{fa}}{\lambda_{miss}}\frac{P(N)}{P(T)}$ are known or can be estimated. Alternatively, Inequality 4.1 can be used to motivate a simpler rule. On the right hand side, the first term is a constant. The second term is inversely related to the number of observations, therefore its influence decreases with increasing number of observations. Consequently, for large n, the inequality may be

approximated with a constant, which suggests the decision rule to label a state as a target if (1) the number of observations on this state is above a threshold parameter t_n, and (2) the proportion of positive observations is greater than another threshold parameter t_p:

$$n > t_n \quad \text{and} \quad \frac{n_+}{n} > t_p. \tag{4.2}$$

To arrive at this decision rule, we made two simplifying assumptions that are likely to be violated to some degree. First, placing all states in one of two classes is a simplification. The probability of producing a positive observation may vary considerably within states of the same class. Second, if local interaction graphs are constructed from overlapping trajectories, observations on a given state are not entirely independent. Nevertheless, the simple and intuitive decision rules obtained with these assumptions are surprisingly effective in identifying access states.

4.2.2 The Local Betweenness Algorithm (LoBet)

The formulation of the discovery problem as a classification task suggests an incremental algorithm presented in Figure 4.3. The agent continuously interacts with its environment, constructing local interaction graphs from short, non-overlapping trajectories. From each local interaction graph, it obtains an observation for each state represented on the graph. This is a positive observation if the state is a local maximum of betweenness on the graph, a negative observation otherwise. With each new observation on a given state, the agent reapplies Decision Rule 4.1 or 4.2 to the state, classifying it as an access state if the decision rule is satisfied. The agent continues this incremental process indefinitely. I call this algorithm the *Local Betweenness Algorithm* (LoBet).

Each local interaction graph may be processed in time $O(l^2)$ if edge weights are uniform, in time $O(l^2 + l^2 \log l)$ otherwise. The parameter l, the length of the input

Parameters if using Decision Rule 4.1	
l	Trajectory length
p	Probability of a positive observation given an access state
q	Probability of a positive observation given not an access state
$\frac{P(N)}{P(T)}$	Prior ratio
$\frac{\lambda_{fa}}{\lambda_{miss}}$	Cost ratio

Parameters if using Decision Rule 4.2	
l	Trajectory length
t_n	Threshold on required observations on a state
t_p	Threshold on proportion of positive observations

Algorithm

$o_i = 0$, $o_i^+ = 0$
Repeat forever:
  Observe state s
  $V = \{s\}$, $E = \emptyset$, $n_{ij} = 0$, $w_{ij} = 0$
  Repeat l times:
    Take action a, observe reward r and next state s'
    $V = V \cup \{s'\}$
    $E = E \cup (s, s')$
    $n_{ss'} = n_{ss'} + 1$
    $w_{ss'} = w_{ss'} + (r - w_{ss'})/n_{ss'}$
    Compute betweenness on graph $G = (V, E)$ using edge costs $-w_{ij}$.
    For all $s \in V$:
      $o_s = o_s + 1$
      If s is a local maximum on G: $o_s^+ = o_s^+ + 1$
      If the decision rule is satisfied, classify s as an access state.

Figure 4.3. The Local Betweenness Algorithm (LoBet).

trajectory, is a small constant not directly related to the number of states in the domain.

4.2.3 Performance

Figure 4.4 shows the results of applying the Local Betweenness Algorithm in the Rooms domain while the agent was performing a random walk. The algorithm was applied using Decision Rule 4.1, with $l = 1000$, $\frac{\lambda_{fa}}{\lambda_{miss}} = 100$, $\frac{p(N)}{p(T)} = 100$, $p = 0.36$,

$q = 0.036$. The parameters p and q were estimated from the random walk discussed earlier in Section 4.2.

The figure shows the number of times each state was identified as an access state in 100 trials. The local maxima of betweenness in this domain are the fourteen states that are adjacent to the seven doorways. The doorways themselves have slightly lower betweenness values, therefore they are not local maxima with respect to the definition employed here. These fourteen states were identified as access states in all 100 trials. Very few additional states were identified as access states, amounting to less than one state per trial. Figure 4.4 also shows how the discovery process progressed over time, revealing that the discovery rate eventually reduced to approximately zero rather than indefinitely continuing at a substantial rate.

Figure 4.5 shows similar results in the Shortcuts domain introduced in Section 3.2.2. All experimental conditions were identical to their settings in the Rooms example, except p and q were similarly determined while the agent was performing a random walk ($p = 0.3$, $q = 0.07$). The local maxima of betweenness in this domain are the four states that border the two shortcut edges. All of them were identified as access states in all 100 trials. Few additional states were identified as access states, amounting to less than 0.2 states per trial.

4.3 Local Graph Partitioning

The second method I propose for identifying access states also takes a graph-theoretic approach. In particular, it also operates on local interaction graphs rather than the full interaction graph of the domain. Instead of computing betweenness on these graphs, this alternative approach identifies cuts that partition the graph into two blocks with a low between-blocks transition probability. States that consistently border identified cuts are labeled as access states.

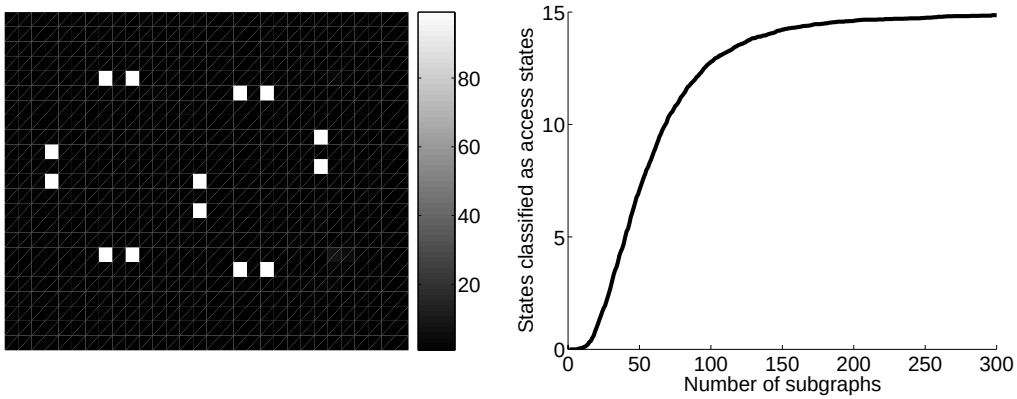


Figure 4.4. Performance of LoBet in the Rooms domain during a random walk.

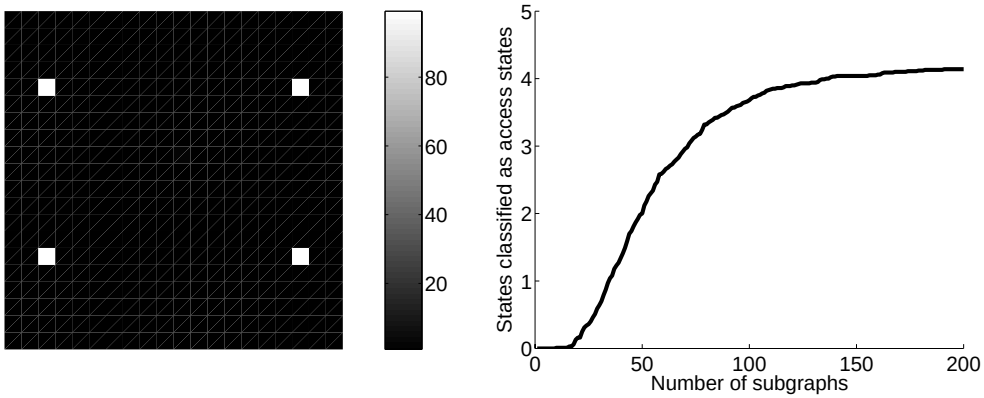


Figure 4.5. Performance of LoBet in the Shortcuts domain during a random walk.

4.3.1 Utility of Local Cuts

The local scope of the interaction graph is a key property of this discovery method. The cuts it performs do not partition the entire state set but only a small part of it encountered in a short, continuous fragment of experience. This local perspective allows the approach to identify a large set of access states that are missed by methods that perform cuts of the full interaction graph, such as the discovery algorithms by Menache et al. (2002) and Mannor et al. (2004).

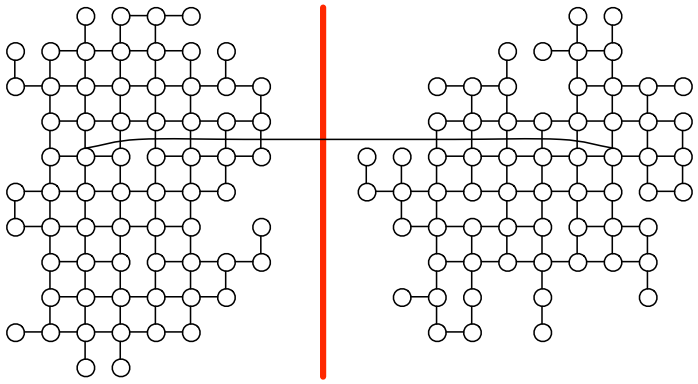


Figure 4.6. A sample local interaction graph in the Shortcuts domain.

The Shortcuts domain discussed earlier in Section 3.2.2 illustrates the difference between global and local cuts. Global cuts miss the access states in this domain as discussed in Section 3.4 and illustrated in Figure 3.14. In contrast, a local cut can easily isolate the access states as illustrated in Figure 4.6. Not all transition sequences yield a graph like the one in this figure, but the classification framework described in Section 4.2.1 can be similarly used here to combine evidence from an ensemble of local interaction graphs to reliably determine access states of the domain.

4.3.2 Cut Metric

Given a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges, a cut (A, B) of G is a partition of V. The edges that cross the cut are those with one endpoint in block A and the other in block B.

A cut metric particularly well suited to identifying access states is the normalized cut ($NCut$) introduced by Shi and Malik (2000):

$$NCut(A, B) = \frac{cut(A, B)}{vol(A)} + \frac{cut(B, A)}{vol(B)}, \tag{4.3}$$

where $cut(A, B)$ is the sum of the weights on edges that originate in A and terminate in B, and $vol(A)$ is the sum of weights on all edges that originate in A.[2] For a local interaction graph with edge weights that denote transition frequencies, the first term in Equation 4.3 is the number of observed transitions from a state in block A to a state in block B divided by the total number of transitions from states in block A. The result is an estimate of the probability that the agent transitions to block B in one step under its current policy given that it starts in block A. A similar argument can be made for the second term in Equation 4.3 for transitioning in the other direction. Their sum, *NCut*, is an estimate of the sum of probabilities of crossing the cut from each block.

Alternative cut metrics commonly used in graph partitioning are *MinCut* (Wu and Leahy 1993) and *RatioCut* (Hagen and Kahng 1992). *MinCut* is the sum of edge weights that cross the cut. *RatioCut* equals $cut(A, B)/|A| + cut(A, B)/|B|$ for an undirected graph, where $|A|$ and $|B|$ are the number of vertices in blocks A and B, respectively. Neither of these metrics are as suitable as *NCut* for identifying access states. *MinCut*, in particular, creates a bias towards cuts that separate a small number of nodes from the rest of the graph, for example a single corner state in a gridworld, and is inferior to the other two metrics for the purpose of identifying access states.

4.3.3 Partitioning Algorithm

Finding a partition of a graph that minimizes *NCut* is NP-hard, but there exist good approximate algorithms. In the experiments reported here, the spectral clustering algorithm of Shi and Malik (2000) is used on an undirected version of the local interaction graph in which edge weights show the number of transitions in either direction. This algorithm has a running time of $O(N^3)$, where N is the number of

[2]The choice of *NCut* as the cut metric is due to Alicia P. Wolfe.

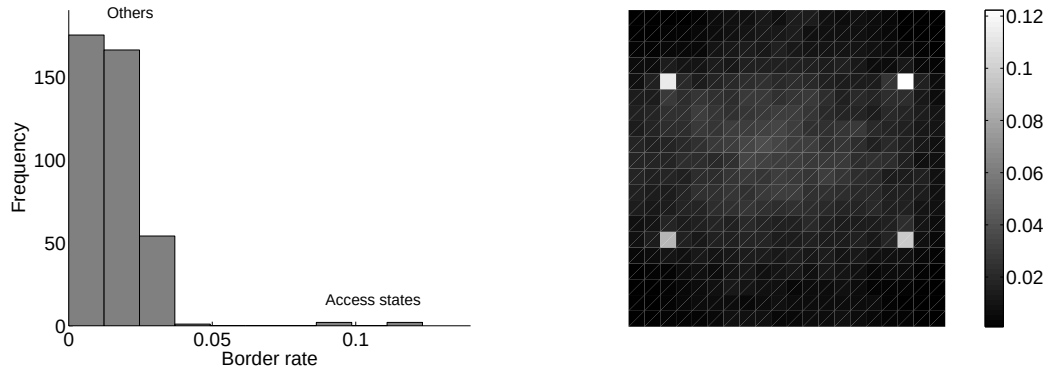


Figure 4.7. Border rates in the Shortcuts domain.

vertices in the graph. Because the approach proposed here works with local interaction graphs rather than the full interaction graph of the domain, N is not directly related to the number of states in the domain.

4.3.4 Local Cuts

I hypothesize that access states are more likely than other states to border cuts of the local interaction graphs in which the blocks are well separated. An analysis performed in the Shortcuts domain supports this hypothesis. The agent performed 10,000 random walks of 500 steps, each starting at a random location. At the end of each random walk, a local interaction graph was constructed that reflected the agent's brief experience in the domain. The mean number of vertices in these graphs was 153. If the cut that minimized *NCut* had a low *NCut* value (less than 0.02), indicating that the blocks were well separated, the states that border the cut edges were labeled as border states.

Figure 4.7 shows a histogram of the border rates obtained in the domain, where *border rate* of a state is the number of times the state was labeled as a border state divided by the number of local interaction graphs on which the state was represented. The figure shows a bimodal distribution, as expected. The four access states of the

domain had border rates above 0.08 while the rest of the states had border rates that were below 0.04. The figure also visually depicts the border rate of each state in the domain, showing clearly the contrast between the four access states and the other states.

4.3.5 Formulation as a Classification Problem

The discrepancy in the border rates of access states and other states in the domain makes it possible to formulate the problem of identifying access states as a classification task, analogous to the formulation presented in Section 4.2.1. This formulation yields exactly the same decision rules as in Section 4.2.1. The only difference here is in how the positive and negative observations are obtained for each state. Local interaction graphs are still the source of the observations, but the processing of these graphs involves graph partitioning rather than computing vertex betweenness. As in the earlier formulation, each local interaction graph yields one observation for each state represented on it. This is a positive observation if the state is a border state on the graph, a negative observation otherwise. Both positive and negative observations may be associated with each state, but access states have a higher probability than other states of producing positive observations.

4.3.6 The Local Cuts Algorithm (L-Cut)

The formulation of the discovery problem as a classification task suggests an incremental algorithm presented in Figure 4.8 that is quite similar to the Local Betweenness Algorithm (LoBet) introduced earlier. The agent continuously interacts with its environment, constructing local interaction graphs from short, non-overlapping trajectories. Each local interaction graph produces a new observation for each state represented on it. A low *NCut* value indicates a good separation between the blocks. In this case, the states that are endpoints of the edges that cross the cut obtain a positive observation while the remaining states obtain a negative observation. On

Parameters if using Decision Rule 4.1	
l	Trajectory length
t_c	Cut threshold
p	Probability of a positive observation given an access state
q	Probability of a positive observation given not an access state
$\frac{P(N)}{P(T)}$	Prior ratio
$\frac{\lambda_{fa}}{\lambda_{miss}}$	Cost ratio

Parameters if using Decision Rule 4.2	
l	Trajectory length
t_c	Cut threshold
t_n	Threshold on required observations on a state
t_p	Threshold on proportion of positive observations

Algorithm

$o_i = 0$, $o_i^+ = 0$
Repeat forever:
  Observe state s
  $V = \{s\}$, $E = \emptyset$, $w_{ij} = 0$
  Repeat l times:
    Take action a, observe reward r and next state s'
    $V = V \cup \{s'\}$
    $E = E \cup (s, s')$
    $w_{ss'} = w_{ss'} + 1$
    Identify the cut (A, B) that minimizes $NCut$ on graph $G = (V, E)$
    For all $s \in V$:
      $o_s = o_s + 1$
      If $NCut(A, B) < t_c$ and s borders (A, B): $o_s^+ = o_s^+ + 1$
      If decision rule is satisfied, classify s as an access state

Figure 4.8. The Local Cuts Algorithm (L-Cut).

the other hand, a high *NCut* value indicates that the blocks are not well separated, in which case all states in the graph obtain a negative observation. The threshold *NCut* value (t_c) is a parameter of the algorithm. With each new observation on a given state, the agent reapplies Decision Rule 4.1 or 4.2 to the state and classifies the state as an access state if the decision rule is satisfied. The agent continues this incremental process indefinitely.

Local interaction graphs are constructed periodically from non-overlapping trajectories of length l, another parameter of the algorithm. Edge weights used in graph partitioning are the number of corresponding transitions that take place in the trajectory.

Each local interaction graph may be processed in time $O(l^3)$. The parameter l, the length of the input trajectory, is a small constant not directly related to the number of states in the domain.

4.3.7 Performance

Figure 4.9 shows the results of applying the Local Cuts Algorithm in the Shortcuts domain while the agent was performing a random walk. The algorithm was applied using Decision Rule 4.1, with $l = 500$, $\frac{\lambda_{fa}}{\lambda_{miss}} = 100$, $\frac{p(N)}{p(T)} = 100$, $p = 0.1053$, $q = 0.0142$. The parameters p and q were estimated from the random walk discussed earlier in Section 4.3.4. The figures show that L-Cut succeeded in isolating the access states of the domain. In all 100 trials, at least one border state of both shortcut edges were identified as an access state. No state other than the four states that border the shortcut edges was identified as an access state in any of the trials.

Figure 4.10 shows similar results in the Rooms domain. All experimental conditions were identical to their earlier settings, except p and q were similarly determined while the agent was performing a random walk ($p = 0.1652$, $q = 0.0065$). The algorithm was successful in this domain as well. In all 100 trials, the seven doorways

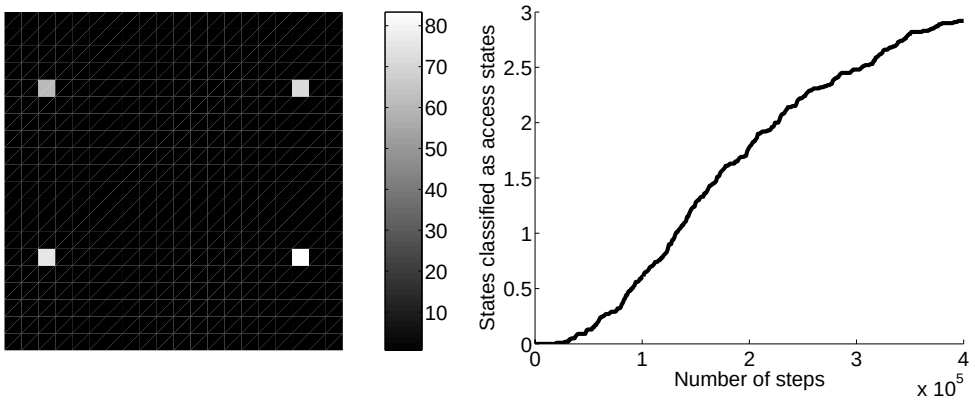


Figure 4.9. Performance of L-Cut in the Shortcuts domain during a random walk.

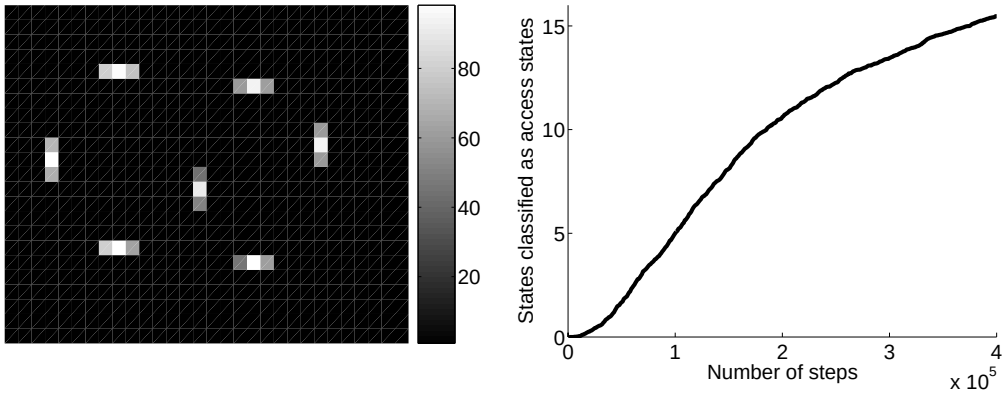


Figure 4.10. Performance of L-Cut in the Rooms domain during a random walk.

or their immediate neighbors were identified as access states. All states that were labeled as access states were states that are within 2 transitions of the doorways.

4.4 Relative Novelty

Unlike LoBet and L-Cut, the next approach I propose for identifying access states does not use the graphical structure of the domain directly. This third approach is founded on the observation that access states are likely to allow the agent to transition to a different region in the state space. To detect such transitions, I introduce the con-

cept of *relative novelty*, a measure of how much short-term novelty a state introduces to the agent. Below, I define novelty and relative novelty, formulate the discovery problem as a classification task analogous to the earlier formulations, describe the full algorithm, and show examples of its behavior in sample domains.

4.4.1 Novelty

Various concepts of novelty play many roles in both cognitive and computational science. I purposefully use a simple notion of novelty that can later be revised for richer formalisms than the discrete state problems that are addressed here. This definition of novelty makes use of the number of visits since a designated start time. The *novelty* of a set of states equals $n^{-1/k}$, where n is the mean number of visits to states in this set and $k > 0$ is a parameter. With this definition, the novelty of a single state equals 1 when it is first visited, decays with each succeeding visit, and approaches 0 in the limit.

4.4.2 Relative Novelty

The *relative novelty* of a state in a transition sequence is the ratio of the novelty of states that followed it (including itself) to the novelty of the states that preceded it. The number of forward and backward transitions to take into account in computing this score is a parameter called the *novelty lag* (l_n). A state is likely to produce a different relative novelty score each time it is visited.

Figure 4.11 shows the distribution of relative novelty scores in the Rooms domain during a 1000-step random walk repeated 10,000 times, starting each at a random grid location. Relative novelty was computed using $k = 2$ and $l_n = 7$. The figure shows the distribution of relative novelty scores separately for access states (seven doorways and fourteen states adjacent to them) and for other states, revealing a difference between the two distributions. Both distributions peak around a relative novelty score of 1,

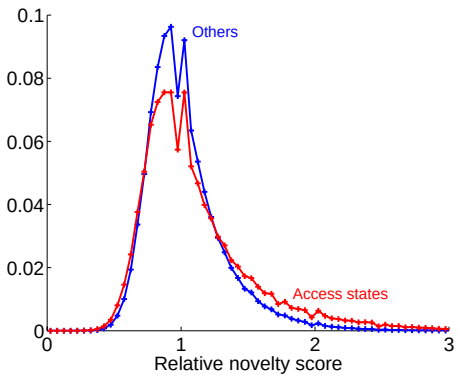


Figure 4.11. Empirical probability distribution function of relative novelty scores in the Rooms domain.

indicating approximately equal novelty scores preceding and following a state, but the distribution for access states has a heavier tail.

A state is defined to introduce relative novelty when its relative novelty score is greater than a parameter called the *relative novelty threshold* (t_{RN}). Figure 4.12 shows the *relative-novelty rate*, the proportion of relative novelty scores that are greater than the threshold, when t_{RN} was set to 2. The conversion from a continuous relative novelty score to a binary feature adequately captures the differences among states because the distributions of relative novelty scores differ mainly in their tail. The figure shows clearly that the access states introduced relative novelty at a higher rate than the other states in the domain.

Repeated experiments with different settings of parameters k and l showed a similar discrepancy in relative novelty scores of the access states and other states in the domain. This discrepancy is the basis of the algorithm I propose.

4.4.3 Formulation as a Classification Problem

The problem posed by the agent may once again be formulated as a classification task, as in earlier approaches, with the premise that access states have a higher probability of producing positive observations than other states. In this case, the binary

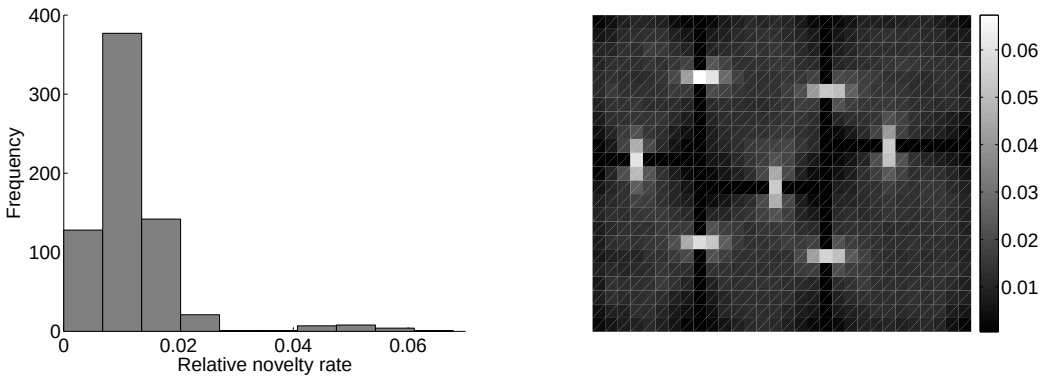


Figure 4.12. Relative novelty rates in the Rooms domain.

observations indicate whether a particular visit to a state introduced relative novelty. Observations are collected continuously as the agent interacts with its environment. With each transition, the agent obtains a new observation for some state s and wishes to determine whether s is an access state based on all observations obtained so far for s. The formulation in Section 4.2.1 applies exactly, except for how the observations are obtained, yielding the two decision rules used earlier for LoBet and L-Cut.

4.4.4 The Relative Novelty Algorithm (RN)

The formulation of the discovery problem as a classification task suggests an incremental algorithm similar to LoBet and L-Cut. This algorithm is presented in Figure 4.13. The algorithm computes a new relative novelty score for a given state each time this state is visited. If this score is greater than the relative novelty threshold, it produces a *positive* observation for the state, otherwise a *negative* observation. Given the number of observations so far accumulated on this state, Decision Rule 4.1 or 4.2 is then used to determine whether to label it as an access state. The agent continues this incremental process indefinitely.

There are some details to this relatively simple procedure. First, it is essential to periodically reset visitation counts. It is important to do so because the type of

Parameters if using Decision Rule 4.1	
k	Novelty exponent
l_n	Novelty lag
t_{RN}	Relative novelty threshold
p	Probability of a positive observation given an access state
q	Probability of a positive observation given not an access state
$\frac{P(N)}{P(T)}$	Prior ratio
$\frac{\lambda_{fa}}{\lambda_{miss}}$	Cost ratio

Parameters if using Decision Rule 4.2	
k	Novelty exponent
l_n	Novelty lag
t_{RN}	Relative novelty threshold
t_n	Threshold on required observations on a state
t_p	Threshold on proportion of positive observations

Algorithm

$o_i = 0$, $o_i^+ = 0$
Repeat at each decision stage t:
  s = state visited at decision stage $t - l_n + 1$
  If $s!$ = state visited at decision stage $t - l_n$:
    RN=relative novelty score for decision stage $t - l_n + 1$
    $o_s = o_s + 1$
    If $RN > t_{RN}$: $o_s^+ = o_s^+ + 1$
    If decision rule is satisfied, classify s as an access state.

Figure 4.13. The Relative Novelty Algorithm (RN).

novelty sought here is defined *relative to the agent's recent experience*. It is irrelevant whether a state is novel to the agent overall. Second, self-transitions (transitions from a state to itself) are ignored because two consecutive scores for the same state are highly correlated, violating the assumption of independent observations in the formulation of the problem as a classification task. And third, there is a time lag between the actual state visitation and the relative novelty computations because of the novelty lag.

RN has very low computational complexity. Each new experience sample may be processed in time $O(1)$.

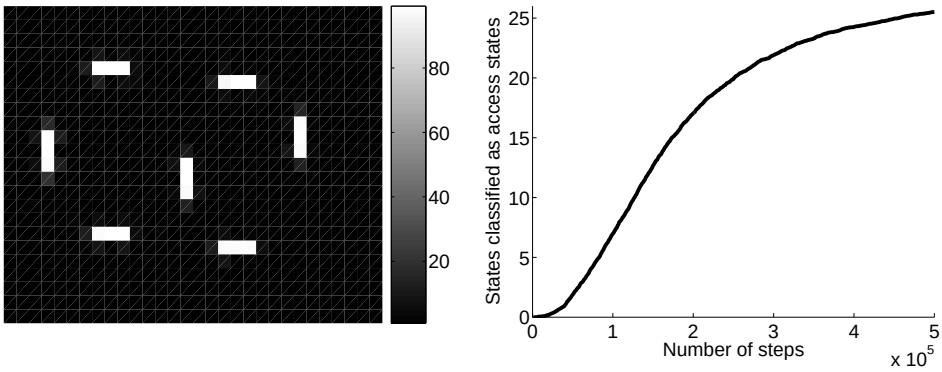


Figure 4.14. Performance of RN in the Rooms domain during a random walk.

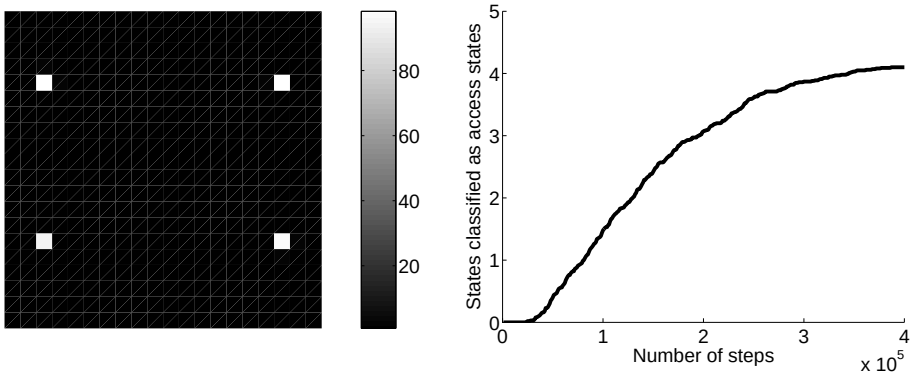


Figure 4.15. Performance of RN in the Shortcuts domain during a random walk.

4.4.5 Performance

Figure 4.14 shows the results of applying the Relative Novelty Algorithm in the Rooms domain while the agent was performing a random walk. The algorithm was applied using Decision Rule 4.1, with $k = 2$, $t_{RN} = 2$, $\frac{\lambda_{fa}}{\lambda_{miss}} = 100$, $\frac{p(N)}{p(T)} = 100$, $p = 0.05$, $q = 0.01$. The parameters p and q were estimated from the random walk discussed earlier in Section 4.4. The figure shows the number of times each state was identified as an access state in 100 trials as well as how the discovery process progressed over time. The results are similar to those obtained with earlier algorithms. RN succeeded in identifying the access states of the domain, labeling

very few additional states as access states. Figure 4.15 shows similar results in the Shortcuts domain under identical experimental conditions.

4.5 Sensitivity Analysis

LoBet, L-Cut, and RN are inspired by the same statistical formulation of the discovery problem. They use the same decision rule to distinguish access states from the other states in the domain, differing only on the type of observations they collect. Here, I investigate the sensitivity of their decision rule to its parameters: $\frac{P(N)}{P(T)}$, $\frac{\lambda_{fa}}{\lambda_{miss}}$, p, and q.

In general, the more separation between the class-conditional probabilities, the more robust the algorithms are to the settings of the decision-rule parameters. In the examples given in this chapter, the smallest separation between the class-conditional probabilities was observed when using RN in the Rooms domain. The sensitivity analysis was conducted in this setting.

4.5.1 Priors and Misclassification Costs

The priors and the misclassification costs appear together in the decision rule as a product: $\frac{P(N)}{P(T)}\frac{\lambda_{fa}}{\lambda_{miss}}$. Figure 4.16 shows the decision threshold for various settings of the product term $\frac{P(N)}{P(T)}\frac{\lambda_{fa}}{\lambda_{miss}}$, ranging from 312 to 20,000, when p and q were kept at their values reported in Section 4.4.5. The product term plays a role when the number of observations is small. Its role in the decision rule is to increase the threshold for low sample sizes when uncertainty about the class label is high. With increasing number of observations, the influence of the product term on the threshold decreases, approaching zero in the limit.

Figure 4.17 shows performance results in the Rooms domain with various settings of the product term. The figure shows that access states were correctly labeled at all settings of the product term, but other states were sometimes incorrectly labeled

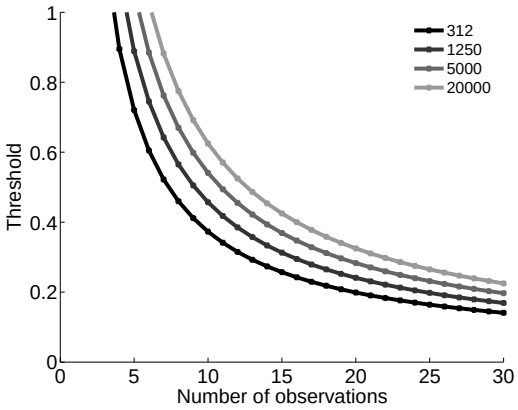


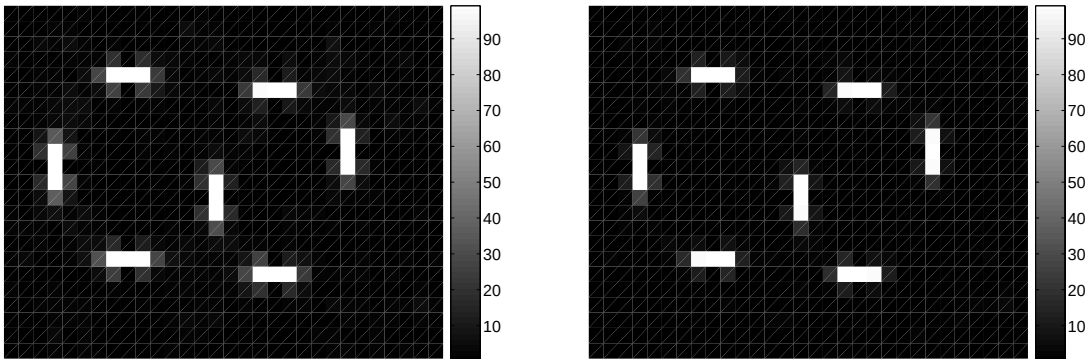
Figure 4.16. Decision threshold for various settings of $\frac{P(N)}{P(T)}\frac{\lambda_{fa}}{\lambda_{miss}}$. Other parameter settings were as reported in Section 4.4.5.

as access states. The number of states labeled incorrectly increased with decreasing values of the product term. In other words, there were no misses, but increasingly more false alarms with decreasing values of the product term. False alarms were not uniformly distributed in the domain, but tended to be close to the access states. As skill subgoals, such states are expected to be more useful than states randomly selected from the domain.

Most false alarms happen early in the discovery process when the number of observations is low. With increasing number of observations, the difference between the decision thresholds with various settings of the product term decreases, resulting in more accurate decisions. Early misclassifications are not a big problem because a skill may be eliminated if further evidence suggests that its subgoal is not an access state.

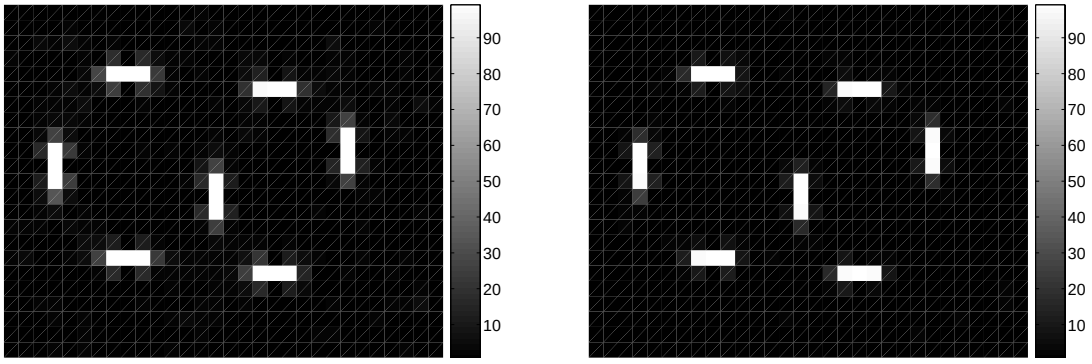
4.5.2 Class-Conditional Probabilities

Figures 4.18 and 4.19 show performance results with various settings of the class-conditional probabilities p and q. The figures show that the degradation in performance is gradual over a range of settings of these parameters. Errors tend to be either



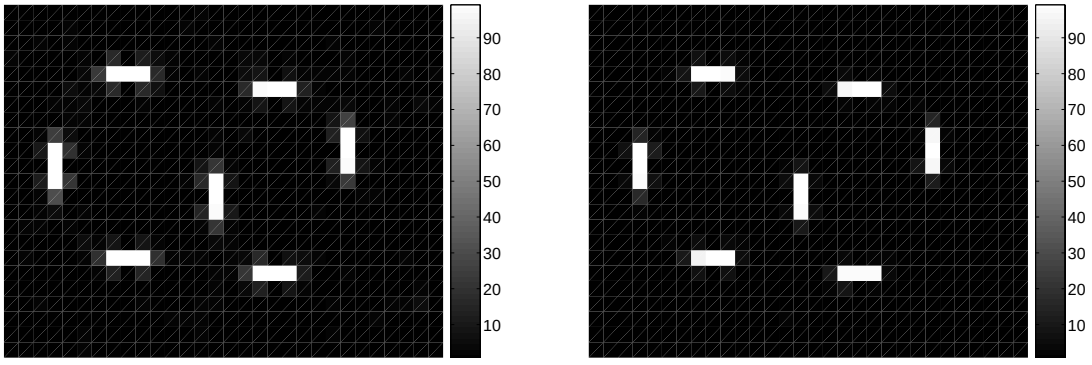
$\frac{P(N)}{P(T)} \frac{\lambda_{fa}}{\lambda_{miss}} = 312$

$\frac{P(N)}{P(T)} \frac{\lambda_{fa}}{\lambda_{miss}} = 2500$



$\frac{P(N)}{P(T)} \frac{\lambda_{fa}}{\lambda_{miss}} = 625$

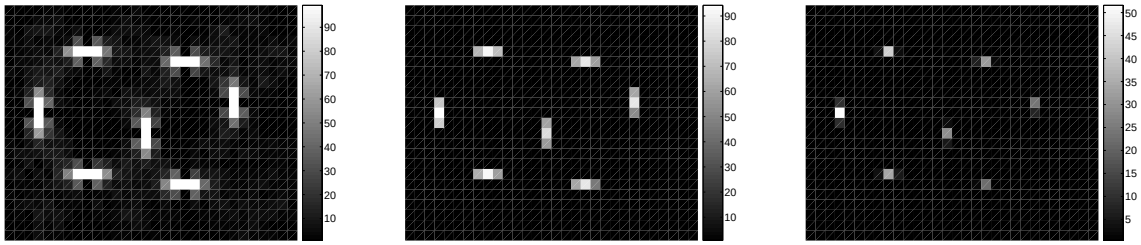
$\frac{P(N)}{P(T)} \frac{\lambda_{fa}}{\lambda_{miss}} = 5000$



$\frac{P(N)}{P(T)} \frac{\lambda_{fa}}{\lambda_{miss}} = 1250$

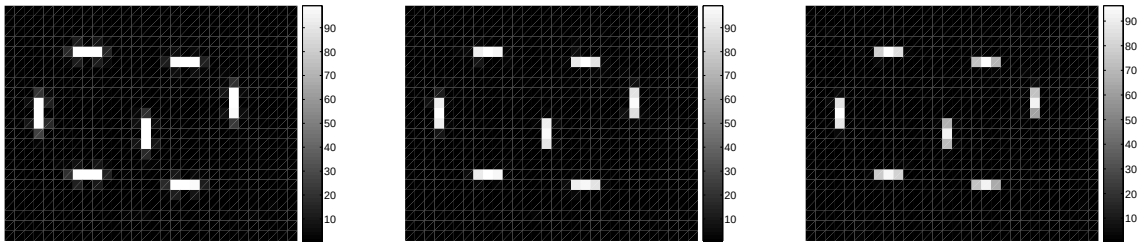
$\frac{P(N)}{P(T)} \frac{\lambda_{fa}}{\lambda_{miss}} = 20,000$

Figure 4.17. Performance of RN in the Rooms domain during a random walk with various settings of $\frac{P(N)}{P(T)} \frac{\lambda_{fa}}{\lambda_{miss}}$. Other parameter settings were as reported in Section 4.4.5.



$q = 0.005$ $q = 0.02$ $q = 0.03$

Figure 4.18. Performance of RN in the Rooms domain during a random walk with various settings of q. Other parameter settings were as reported in Section 4.4.5.



$p = 0.025$ $p = 0.075$ $p = 0.1$

Figure 4.19. Performance of RN in the Rooms domain during a random walk with various settings of p. Other parameter settings were as reported in Section 4.4.5.

false alarms or misses, but not both. In other words, if any states are identified as subgoals, they will include access states of the domain. Furthermore, false alarms tend to be close to the access states of the domain.

4.6 Limitations of Local Methods

The three incremental algorithms proposed for identifying access states rely on the assumption that short trajectories in the domain can, in principle, reveal the different regions that access states lie between. In domains in which this assumption does not hold, the proposed algorithms are not likely to be very successful. The game of Tic-Tac-Toe is one such example. In this domain, the trajectories experienced by the agent are chains that do not reveal any local or global structure in the domain.

Unless the agent combines multiple trajectories together to form one local interaction graph, local approaches can not identify the access states in such domains.

The Relative Novelty and the Local Cuts algorithms further rely on the assumption that the access states are artifacts of the graph connectivity structure. This assumption does not always hold. For instance, access states may be artifacts of the reward structure, as illustrated in Figure 4.20. The figure shows Surfaces, a grid-world domain with varying rewards. The lightly colored squares incur a reward of -0.001 for actions that originate in them, while the darker squares incur -0.1. The domain contains no obstacles other than the surrounding walls, but the varying reward structure creates two doorway-like regions that support efficient navigation in the domain. These regions are useful subgoals for reasons similar to why doorways are useful subgoals.

Because the structure in Surfaces is not a result of node connectivity, RN and L-Cut do not succeed in identifying its access states. In 100 random walks of 400,000 steps, RN and L-Cut identified very few subgoals (less than 0.05 subgoal/trial) uniformly distributed in the domain.

Note, however, that although RN and L-Cut do not succeed in this domain while the agent is performing uninformed exploration, such as a random walk, these algorithms may succeed if the agent is following a policy that is somewhat aware of efficient means of navigation in the domain, as would be the case in middle to late stages of learning how to perform a particular task in the domain. In such cases, the skills that are consequently formed may have limited utility in the current task, because the learning has already progressed quite a bit, but the skills would still be useful for addressing future problems in the domain.

The Local Betweenness Algorithm does not share this limitation. It correctly identifies the access states in the domain as shown in Figure 4.21, which shows the

performance of the algorithm during 100 random walks of 400,000 steps. The algorithm was applied using Decision Rule 4.2, with $l = 1000$, $t_n = 10$, $t_p = 0.20$.

Finally, none of the incremental algorithms presented here can directly take into account the path weights w_{st} in Expression 3.1, which defines access states. They all address the case in which the path weights are uniform. The path weights are important in some situations because they bias the betweenness computation towards regions that are important for the agent. For instance, in the Tic-Tac-Toe example given earlier, paths that terminate with a win for the agent was assigned a weight of $+1$ while all other paths were assigned a weight of 0. With uniform path weights, the access skills of the domain include creating forks in favor of the opponent. One potential remedy for this limitation is to develop a two-tiered approach that continuously refines the skill set by eliminating the skills that are not serving the agent's needs.

4.7 Discussion

The sample problems provided here demonstrate that the local maxima of betweenness may reliably be identified using local information. Besides the computational benefits that an incremental discovery approach would bring, this has another important consequence. Incremental algorithms can provide potentially useful skills in a new environment before the agent learns how to achieve any particular task. This property is essential if a discovery method is to be successfully applied to complex real-world tasks.

A fruitful research direction is to develop algorithms that actively explore to discover access states, rather than only passively mining available trajectories. An active approach that has the objective of producing the most informative local experience can drastically reduce the number of experiences required in incremental algorithms.

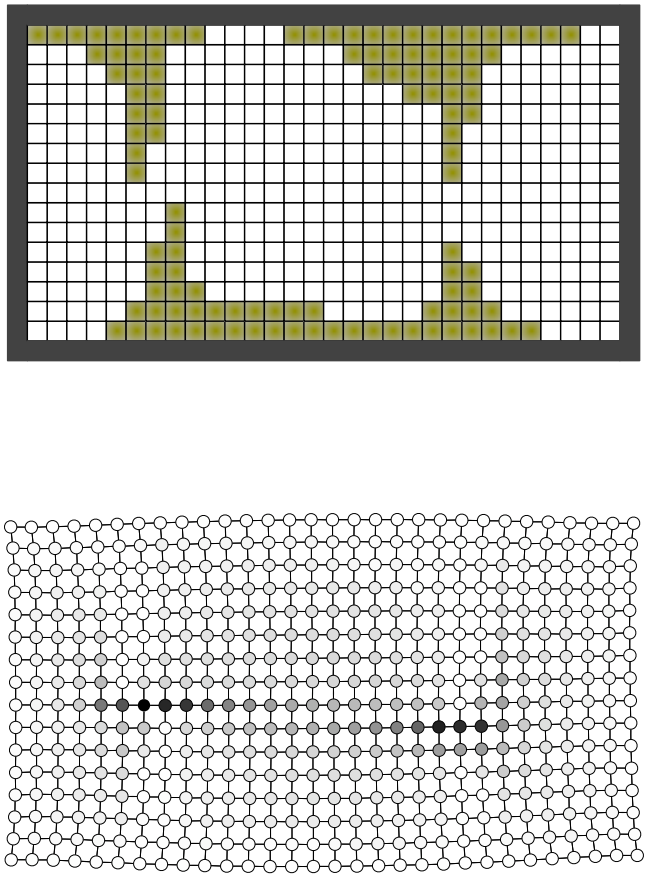


Figure 4.20. The Surfaces domain and its interaction graph. The gray shading on the vertices show betweenness, with black corresponding to the highest betweenness in the domain and white corresponding to the lowest.

Refining the definition of relative novelty for high-dimensional or continuous state spaces is another important direction for future research. The definition used here is for discrete-state problems only and therefore has limited applicability.

Various concepts of novelty are closely linked to motivation and reward in animals (e.g., Kakade & Dayan, 2001). The use of novelty measures to drive the automatic creation of hierarchical behavior architectures may provide useful computational interpretations of novelty-related animal behavior.

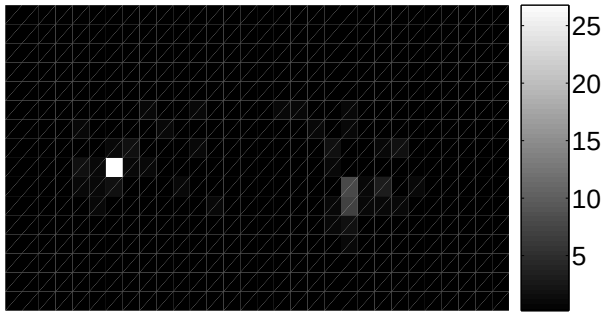


Figure 4.21. Performance of LoBet in the Surfaces domain during a random walk.

4.8 Contributions

The work presented in this chapter demonstrates the utility of the skill characterization presented earlier in informing practical skill-discovery algorithms. The ideas that are introduced here are not developed fully to the extent that they can be immediately applied to practical problems, but they lay the groundwork that future research can successfully build on. Specifically, the work presented in this chapter makes three primary contributions:

It provides evidence that representing or knowing the full interaction graph is not necessary for successfully identifying the local maxima of betweenness. In other words, it demonstrates the feasibility of incremental discovery algorithms for identifying access states.

It introduces three incremental, low-cost approaches for identifying access states that show promise for further development.

It introduces a formulation of the skill-discovery problem as a classification task, deriving simple and effective decision rules for identifying subgoals. This framework may be used to develop alternative discovery algorithms in the future that target access states or an entirely different set of subgoals.

CHAPTER 5

ACQUIRING SKILLS EFFICIENTLY

In this chapter, I develop a class of algorithms for efficiently learning how to reach a desired state. These algorithms specify a behavioral policy in terms of primitive actions and therefore may be considered skills themselves. I call them *trainer skills*.

Trainer skills direct the actions of an agent so that the agent efficiently acquires an optimal policy for later use. That is, an agent executing a trainer skill efficiently learns how to maximize expected return without necessarily collecting high reward during the process. The relevant reward signal may be received from the environment or may be specified by the agent itself.

The objective of a trainer skill is distinct from the traditional objective of a reinforcement learning agent—maximizing return within the agent's lifetime—which requires a trade-off between exploration and exploitation. In contrast, a trainer skill aims to perform *optimal exploration* for the sake of learning a policy that will enable exploitation when needed at a later time.

Trainer skills are useful in a number of contexts. They are a natural fit for problems that can be approached using a two-stage structure—train first, test later. Consider, for example, a team participating in the Robocup Soccer Competition[1]. Goals scored before the competition have no consequence, therefore the best use of this time is to learn how to score goals. The learned policy can then be used during the competition to score as many goals as possible.

[1] http://www.robocup.org/

Trainer skills may also be useful in addressing the traditional reinforcement learning problem. In this context, a trainer skill may be executed as a multi-step exploration policy, for example, instead of the one-step exploratory actions taken by the widely-used ϵ-greedy method.

Finally, trainer skills can be used to efficiently acquire policies for other skills. Most skill discovery methods first identify a reward function that the skill should maximize, then learn a corresponding policy. The latter, if done during a period devoted exclusively to learning a satisfactory skill policy, is an instance of the problem addressed by trainer skills. Such an active approach to skill acquisition has not been discussed in the literature, but a short-term indifference to reward accumulation may prove to be beneficial in this context.

To derive a policy to be followed by trainer skills, I assume that the agent faces an MDP and refer to it as the *task MDP*. I formulate the optimal exploration problem as a different MDP, which I call the *derived MDP*. The states of the derived MDP have two components: an external state that designates the state of the task MDP and an internal state that refers to the internal data structures of the agent. When the agent acts according to the optimal policy of the derived MDP, it performs optimal exploration for learning an optimal policy of the task MDP. This formulation is adapted from that of Duff (2003), where the internal state of the agent is a probability distribution over possible models of the task MDP. Instead of optimally exploring for the sake of identifying the task MDP as in Duff's work, the objective of a trainer skill is to optimally explore to form an optimal policy for the task MDP.

I propose an approximate solution to the derived MDP that produces a simple and intuitive algorithm for specifying the policy of a trainer skill. The algorithm is schematically represented in Figure 5.1 when the objective is to maximize a reward signal obtained from the environment. The figure shows two value functions, a *task value function* that can be used to maximize external reward (in the future) and a

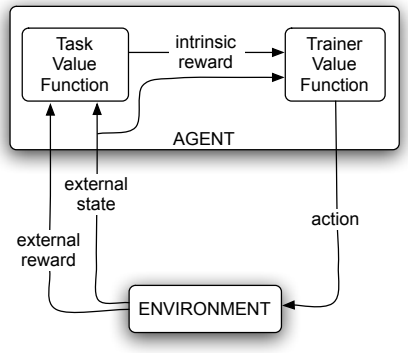


Figure 5.1. A schematic representation of my approach. External state and reward are used to update the task value function. This update produces an intrinsic reward that is used to update the trainer value function.

trainer value function that is used to select actions in the present. The task value function is updated in the usual manner using external state and reward. The updates to the trainer value function ignore external reward but use a different reward signal that depends on the evolution of the task value function. Because this reward signal is a function of the internal state of the agent, it is an *intrinsic* reward as defined by Barto et al. (2004) and Singh et al. (2005).

In the following sections, I first define the optimal exploration problem, formulate it as an MDP, and describe the solution I propose. I then illustrate the use of trainer skills in a number of learning problems and conclude with a discussion of related and future work.[2]

5.1 Optimal Exploration Problem

The optimal exploration problem is to devise an action selection mechanism for generating trajectories in the task MDP such that the policy learned by the end of

[2]The ideas presented in this chapter have appeared in Şimşek and Barto (2006).

a given number of training experiences has as high a value as possible as defined by Equation 2.1.

I assume that the learning algorithm maintains a value function, but otherwise I treat it as a black box, seeking to devise a method that will adapt to the particular algorithm being used. I assume that the structure of the task MDP is unknown, and, furthermore, that it is not possible to sample a transition from an arbitrary state but only from its current state.

5.2 Formulation as an MDP

In the context described above, an action has two direct consequences. First, it generates a training experience for learning the task MDP by revealing an immediate reward and the next state. Second, it changes the environment state, determining which external state will be sampled next. We can explicitly consider the impact of both on future updates to the value function by modeling the agent's internal state, which consists of the data structures representing the task value function and other relevant entities, in addition to the external state of the environment.

With an appropriate representation of internal state, the joint evolution of internal and external state satisfies the Markov property. Consequently, the optimal exploration problem may be formulated as an MDP each of whose states has two components: external state (s_e) and internal state (s_i). The actions in the derived MDP and their effect on the external state component are identical to those in the task MDP. The internal state includes the policy derived from the agent's current value function for the task MDP, denoted π_{s_i}, and all other information that may impact changes to this in the future. The exact representation of internal state and the transition dynamics of the derived MDP depend on the algorithm used to learn the task MDP value function.

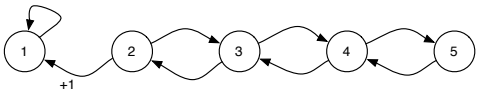


Figure 5.2. A deterministic MDP with five states.

To make matters concrete, consider the deterministic MDP shown in Figure 5.2. All transitions yield zero reward, except for transitions from state 2 into state 1, which yield a reward of +1. The initial state is state 5 with probability 1. Assume the agent uses Q-learning with step size $\alpha = 1$, $\gamma < 1$, and initial Q-values set to zero. Assume also that when the agent reaches the absorbing state, the environment is initialized to the start state (state 5). In this context, it is adequate to consider the agent's internal state to be its current greedy policy. Figure 5.3 shows the state transition dynamics of the derived MDP, depicting an internal state with the corresponding greedy policy for the task MDP. As the policy for the task MDP is improved, transitions move the state of the derived MDP toward the bottom of the diagram. Note that more than one action may be greedy in a given state and that external state 1 is not part of the derived MDP because the agent makes no decisions at this state.

The reward for the derived MDP obtained upon a transition from state (s_e, s_i) to (s'_e, s'_i) is the difference between the values of their associated policies for the task MDP: $V(\pi_{s'_i}) - V(\pi_{s_i})$. It follows that, with $\gamma = 1$, the return obtained in a trajectory of finite length is the difference between the values of policies associated with the last and the first state in the trajectory. Consequently, with $\gamma = 1$ and a horizon that equals the number of training experiences available, the optimal solution to the derived MDP specifies an optimal exploration policy—one that yields a policy with as high a policy value as possible after the specified number of transitions.

It is worthwhile to make a few observations here on the transition structure of the derived MDP. We do not make any assumptions about the structure of the task

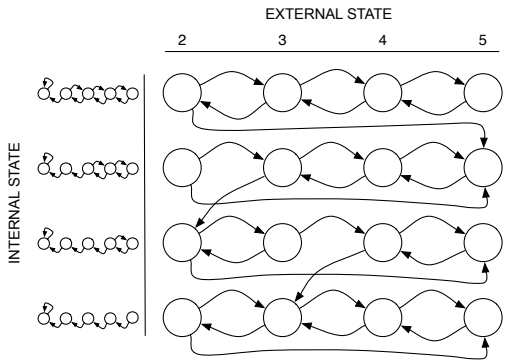


Figure 5.3. State transition graph of the derived MDP corresponding to the task MDP of Figure 5.2. The horizontal axis shows the external state while the vertical axis shows the internal state, depicting an internal state with the associated greedy policy for the task MDP.

MDP, so the transitions along the external state component can be arbitrary. But, transitions along the internal state component have a particular structure. Internal state changes very little from one decision stage to the next because a single training experience changes the value function only slightly. Furthermore, in general (but not always), the policy value increases with more experiences. As a consequence, rather than jumping arbitrarily along the internal state dimension, the agent goes through a progression of internal states that typically increase in value.

Furthermore, if we refer to a set of states with the same internal state component as a *layer*, we observe that the connectivity and reward structure of layers that are directly connected are very similar. This property is due to the incremental nature of learning updates. If an external state transition produces a change in the value function, it is likely to produce a similar change the next time it is experienced. In addition, transitions that are close to such transitions are likely to soon produce changes themselves because the changes in the value function propagate in the state space.

5.3 An Approximate Solution

The derived MDP models the agent's learning process and its optimal policy specifies an optimal exploration policy for learning to solve the task MDP. It is, however, not practical to solve the derived MDP exactly. Here, I enumerate the major difficulties and explain how I address them to derive a principled heuristic.

1. *The agent cannot generate simulated experience for learning trials.* The transition probabilities of both the task MDP and the derived MDP are unknown. As a consequence, the only experiences available to solve the derived MDP are those obtained during the training period itself—which the derived MDP is supposed to optimize! Unless one takes a Bayesian approach to explicitly represent the uncertainty in the transition probabilities, which is intractable even for very small problems, the only viable alternative is to learn to solve both MDPs simultaneously, using the current solution to the derived MDP to select actions. This approach is the one I take.

2. *The state set of the derived MDP is enormous*, even if one could easily identify an appropriate internal state representation. In any problem of reasonable size, the agent is unlikely to observe a state of the derived MDP more than once. I address this issue by using state approximation, more specifically by ignoring the internal state component and learning a behavior policy as a function of external state only. When the internal state component is hidden, the derived MDP appears as a non-stationary MDP, with expected reward associated with transitions varying over time as the agent jumps from one layer of the derived MDP to another. This non-stationarity, however, is slowly varying because the directly connected layers are very similar. It should therefore be possible to "track" this slowly-varying non-stationary MDP in the sense of maintaining a nearly optimal (or at least, good) policy over time.

3. *Reward for the derived MDP cannot be computed exactly.* Recall that the reward for transitioning from state (s_e, s_i) to state (s'_e, s'_i) is $V(\pi_{s'_i}) - V(\pi_{s_i})$, or equivalently

$$\sum_{s \in S_e} D(s)\left(V^{\pi_{s'_i}}(s) - V^{\pi_{s_i}}(s)\right), \tag{5.1}$$

which we obtain using Equation 2.1 and where S_e is the state set of the task MDP. This expression is a weighted sum, over all external states, of the change in actual state value from one decision stage to the next, brought about by the updates to the value function of the task MDP. It can not be computed exactly because the actual state values are unknown, but a reasonable estimate can be obtained using the evolution of estimated state values, $V_t(s)$.

To estimate reward, I distinguish between two types of updates to the value function: those that use the new training experience directly as a new sample, for example updates performed by Q-learning with or without eligibility traces, and those that propagate the direct updates in the state space, for example model-based planning updates of Dyna (Sutton 1990). When both updates are present, the reward estimate is

$$\sum_{s \in S_e} D(s)(V_t(s) - V_{t-1}(s)), \tag{5.2}$$

where the transition takes place from decision stage $t-1$ to t. This estimate assumes that the change in estimated value of a state equals the change in its actual value. When only direct updates are present, it is not clear how they would propagate in the state space. The reward estimate in this case is

$$\sum_{s \in S_e} (V_t(s) - V_{t-1}(s)), \tag{5.3}$$

which is independent of the initial state distribution. One can view this estimate as the result of assuming the extreme case that a direct update would propagate undiminished to all the other states; in other words that it would change the estimated value of all other states by the same amount.

With the approximation to the reward function given by Expression 5.2 or 5.3, there are two issues to consider. First, for a given state s, $V_t(s)$ may show high

fluctuations over time. It may therefore be desirable to use a smoother estimate of state value using the history of value functions, for example a moving average or the maximum estimate over history. Second, it is important to have pessimistic initial values because the underlying assumption is that increases in estimated value reflect increases in actual value.

5.4 The Policy of a Trainer Skill

The ideas in the preceding sections produce a simple and intuitive algorithm for specifying the policy of a trainer skill schematically represented in Figure 5.1. The figure shows two value functions: one that can be used to solve the task MDP (in the future) and another that is used to select actions in the present. I call these the *task value function* and the *trainer value function*, respectively, and I call their associated policies the *task policy* and the *trainer policy*.

At decision stage t, the agent executes an action, observes an immediate external reward and the next external state, and updates the task value function. This update produces an intrinsic reward, $r_i(t)$, which the agent uses, together with observed external state, to update the trainer value function. The reinforcement learning algorithm used to update the trainer value function may be different than the algorithm used for learning the task value function. The number of available training experiences does not influence the algorithm, so it does not need to be known in advance. I refer to this algorithm as ΔV.

In the experiments presented here, intrinsic reward was defined as follows:

$$r_i(t) = p + \sum_{s \in S} \left(V_t^{max}(s) - V_{t-1}^{max}(s) \right), \tag{5.4}$$

where $V_t^{max}(s) = \max_{T \leq t} V_T(s)$ and $p < 0$ is a small action penalty. The action penalty does not change the optimal policy of the derived MDP, but tends to promote faster

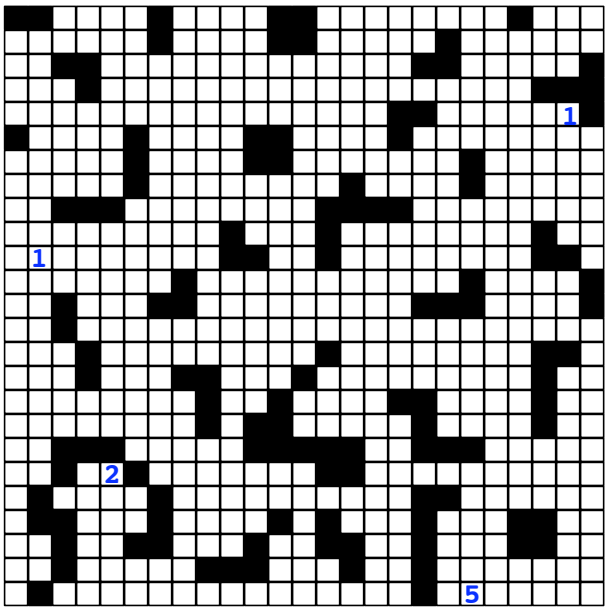


Figure 5.4. The maze task. Terminal states are marked with the amount of reward they generate.

learning and therefore better tracking of the non-stationary MDP observed when the internal state component is ignored.

5.5 Example: Learning to Solve a Maze Task

In this section, I evaluate ΔV in the stochastic maze task shown in Figure 5.4. The available actions in each state are `north`, `south`, `east`, and `west`. These move the agent in the intended direction with probability 0.9 and in a uniform random direction with probability 0.1. If the direction of movement is blocked, the agent remains in the same location. Reward is -0.001 for each action and an additional $+1$, $+2$, or $+5$ when transitioning into one of the terminal states. The start state is the square in the center of the grid with probability 1. When generating training experiences, the agent returned to the start state after reaching a terminal state. The agent used Q-learning with $\alpha = 0.1$ and $\gamma = 0.99$ to learn both the task value function and the

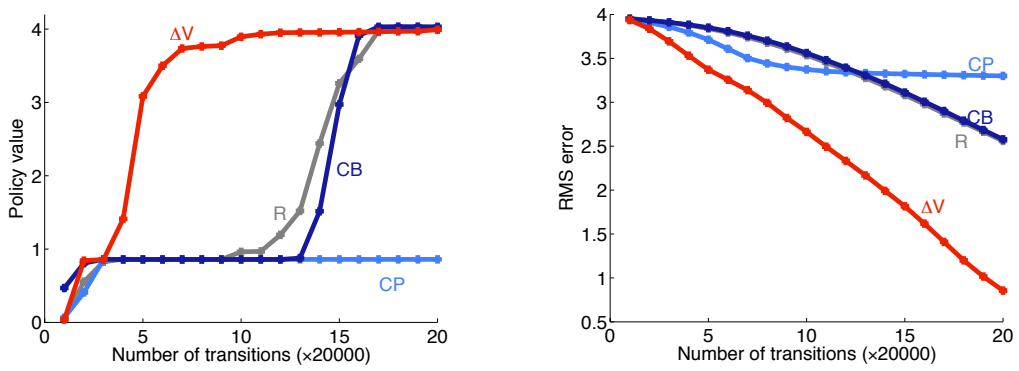


Figure 5.5. Performance in the maze task: (a) Policy value as defined by Equation 2.1, (b) RMS error between the current and optimal state values.

trainer value function. Initial Q-values were zero. The trainer policy was the greedy policy with respect to the trainer value function. The penalty term p was -0.005.

Figure 5.5 shows performance on this task in comparison to a number of baselines. *Random (R)* picked actions uniformly randomly. *Counter-based (CB)* picked the action selected the least number of times from the current state, which is a model-free variant of the action selection mechanism in Thrun (1992). *Constant-Penalty (CP)* was identical to ΔV, but as intrinsic reward used only the penalty term p instead of the right hand side of Equation 5.4.

The figure shows two performance measures: (1) policy value (of the greedy policy with respect to the task value function) computed using Equation 2.1, and (2) root mean-squared (RMS) error between the current and optimal state values. The figure shows means of 30 trials. In both performance measures, ΔV showed clear performance gains over the baselines. These results were typical in a variety of maze tasks.

A closer examination of learning trials revealed that the behavior of ΔV was qualitatively different than the other algorithms, as expected. Its behavior was neither random, nor could it be characterized as repeated systematic sweeps of the state

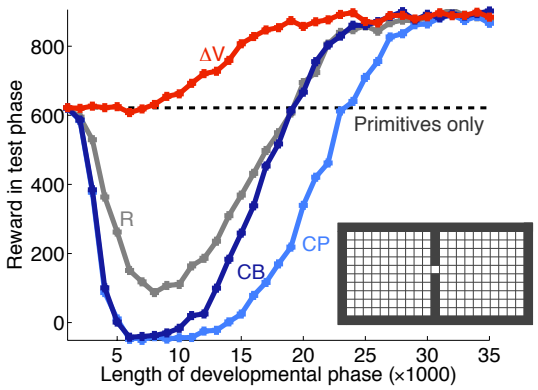


Figure 5.6. Performance in the Rooms skill-acquisition task.

space. Instead, it obsessively remained in regions in which the task value function was improving, efficiently backing up the rewards in the terminal states to the rest of the state space.

5.6 Example: Learning an Approach Skill

The majority of skill discovery methods proceed by generating a reward function that the skill should maximize, typically by identifying a set of states that are useful to reach and defining a skill reward function whose optimal policy efficiently takes the agent to these states, e.g., Hengst (2002), McGovern and Barto (2001). When a new skill reward function is identified, a trainer skill may be used to generate the future experiences of the agent until a satisfactory skill policy is acquired. In this context, a brief period of indifference to external reward may be a small sacrifice with a high payoff. The sooner the skill is functional, the sooner the agent can start obtaining its benefits.

5.6.1 Rooms

I evaluated the utility of ΔV in skill acquisition in the two-room gridworld environment shown in Figure 5.6, with dynamics identical to the maze domain presented

above. A useful skill in this domain, for solving a number of problems, is one that takes the agent efficiently to the doorway. Many existing discovery methods can identify this skill, but, rather than using one of these algorithms, I isolate the exploration problem from the discovery problem by assuming that the agent has at its disposal an ideal discovery method—one that would identify the doorway as a useful subgoal the first time it is experienced.

The experimental method consisted of two phases: a developmental phase in which the agent acquired the skill policy and a test phase in which the agent used the acquired skill (in addition to the primitive actions) to maximize an external reward signal. Performance measure was the total discounted reward obtained in the test phase, during which the agent repeatedly performed an episodic task that started from a random state in the west room and terminated at the southeast corner of the grid. Reward associated with each transition was -0.001, plus an additional $+1$ if the transition took the agent to the terminal state. The skill was made available only from the west room to be able to attribute performance differences only to differences in the quality of the acquired skill policy. Otherwise, poor performance may also be attributed to a well-acquired skill diverting the agent from reaching the terminal state.

When the agent observed the doorway state for the first time, it created an option that terminated with probability 1 at the doorway, with an initiation set containing only the state visited prior to visiting the doorway. The initiation set was expanded with subsequent experiences. Each time the agent transitioned to a state in the initiation set from a state s of the west room outside the initiation set, s was added to the initiation set. The skill reward function assigned -0.001 for each transition and an additional $+1$ for transitioning to the doorway.

In the test phase, the agent used intra-option Q-learning with $\alpha = 0.1$, $\epsilon = 0.05$, $\gamma = 1$. The test phase was 60,000 steps, which was the number of transitions required,

on average, for an agent using only primitive actions to converge to its maximum performance. The algorithms used in the developmental phase and any unspecified parameters were identical to those in the maze task discussed in Section 5.5.

Figure 5.6 shows the total reward obtained in the test phase as a function of the length of the developmental phase measured in number of transitions. The figure shows means over 100 trials. In addition to the previous baselines, the figure shows the performance of an agent that used only primitive actions, which is independent of the length of the developmental phase. The figure shows that ΔV not only obtained the maximum performance with much less experience, but also that it never produced a "harmful" skill with which the agent accumulates less reward than it would using only primitive actions.

5.6.2 Playroom

I repeated a similar experiment in the Playroom domain (discussed in Section 3.2.5) with similar results. The skill learned in this domain was *turning the music on*. The policy acquired for the skill was tested in a collection of randomly-selected tasks in the domain all of which required the agent to turn the music on in order to reach the goal state. All other experimental conditions were identical to those in the grid-world example. The results, shown in Figure 5.7, were qualitatively similar to those obtained in the gridworld skill.

5.7 Discussion

Exploration in reinforcement learning has been studied extensively but typically with the objective of maximizing return in an agent's lifetime, which requires a trade-off between exploration and exploitation, e.g., Duff (2002), Kearns and Singh (1998). Doing so optimally is known as the optimal learning problem. In contrast, the problem addressed by a trainer skill is optimal exploration, in which the objective is to learn

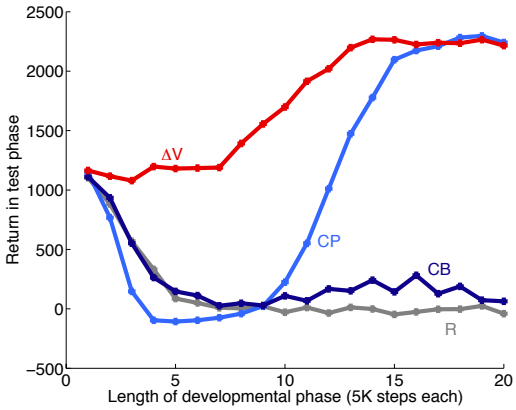


Figure 5.7. Performance in the Playroom skill-acquisition task.

how to maximize return without necessarily accumulating high reward in the process. Despite this difference, the approach taken here adopts aspects of the full Bayesian adaptive approach to solving the optimal learning problem (Duff 2002). In particular, the algorithm is motivated through a derived MDP with states factored into internal and external components analogous to, but not the same as, the information and physical state components of the Bayesian adaptive approach.

The behavior achieved by trainer skills is focused exploration in regions of the state space in which the learning updates improve the task value function the most. A similar behavior is achieved with Prioritized Sweeping (Moore and Atkeson 1993) and Queue-Dyna (Peng and Williams 1993) when a model is used to generate updates to the value function. Theirs is a different problem than the one addressed here because, when a (learned) model is available, states can be selected arbitrarily for backups. Both Prioritized Sweeping and Queue-Dyna are concerned with making the most of available training experience, while trainer skills are concerned with generating the training experience that will be the most useful. As such, Prioritized Sweeping and Queue-Dyna are complementary to the algorithm presented here and can be used in conjunction with it.

An essential component of trainer skills is a reward function defined as a function of the internal state of the agent. Some other work in the literature that does the same are Kaplan and Oudeyer (2003), Schmidhuber (1991), and Schmidhuber and Storck (1993), which are concerned with learning predictive models of the environment. In contrast, trainer skills learn a value function for maximizing an external reward signal. The perspective taken here may help formalize the intuition behind these related algorithms, although a value function is more suitable for this approach than a model. Because the value of a state is a function of the values of its neighbors, changes in the value of one state propagate throughout the state space, helping to create the structure in the derived MDP that trainer skills exploit. This property, in general, does not hold in the case of learning a model. A model, however, also is typically learned incrementally, so the derived MDP would still have some of the same structure when the goal is to learn a model.

The use of trainer skills in the context of skill acquisition is an *active* approach to skill acquisition. Most skill discovery methods are *passive*, in that they do not direct the agent to seek experiences that will be useful for acquiring the skill. Instead, they use whatever experiences are available. One exception is the algorithm by Barto et al. (2004) and Singh et al. (2005) in which an intrinsic reward term is added to the external reward function. While this method does not create a pure exploration policy as a trainer skill, it has a similar idea of influencing the behavior towards those experiences that would lead to efficient learning of the skill policy. Subsequent research, however, has shown that this method is not effective in achieving such behavior (Barto and Şimşek 2005).

CHAPTER 6

CONTRIBUTIONS

In this dissertation, I have addressed a series of questions on skill acquisition in autonomous agents. This is a subject of fundamental importance in AI. New insights have the potential to significantly extend the capabilities of autonomous agents, allowing them to operate at a qualitatively different level than the agents of today.

I started with a conceptual question: What constitutes a useful skill? I presented one answer to this question using the properties of the graphical representation of the agent's interaction with its environment as a common language across different problems. In a wide range of problems, my answer captures a broad range of skills that are intuitively appealing, that correspond to what people handcraft for these problems, and that improve empirical learning performance. In the game of Tic-Tac-Toe, these skills set up a fork to force the opponent to lose. In the Towers of Hanoi puzzle, they include clearing the stack on top of the largest disk, as well as clearing another peg entirely, to be able to move the largest disk to another peg.

The skill characterization I presented captures and generalizes—at least intuitively—the "bottleneck" idea, an intuitive concept that has inspired many existing skill discovery algorithms. By concretely defining what makes a useful skill, my characterization illuminates the limitations of existing algorithms in terms of the type of bottlenecks they miss. My characterization also suggests that the discovery process may succeed with much less sample and computational complexity than required by existing algorithms.

The second question I addressed was algorithmic: How can an agent autonomously identify such skills? The skill characterization I presented can be used directly as a discovery algorithm, but its use is limited to the settings in which the graphical representation of the agent's interaction with its environment is known and is small enough that the computational cost is not a burden. In many cases, the interaction graph is not available, but the agent can sample paths from the graph by interacting with its environment, trying out different actions and observing their consequences. For this more challenging but more prevalent setting, I presented three algorithms with low computational complexity that do not require complete representation of the graph. These algorithms succeed in their current form in some simple domains, but need to be further developed to be applicable to large, complex domains with varying graphical structure.

The discovery methods I proposed here address some of the limitations of existing algorithms. Although not comprehensive or conclusive, the analysis I presented suggests that, through further development, the methods I proposed have the potential to identify a broader set of skills than existing algorithms, with less computational and sample requirements. Most importantly, the discovery methods proposed here illustrate the feasibility of acquiring useful skills without the need to explore a large part of the environment or the need to achieve the desired behavior at least once. Most existing algorithms, if not all of them, require one of these conditions to succeed.

Finally, I addressed another algorithmic question: How can an agent autonomously acquire a desired skill? While the earlier question was concerned with what the skill should accomplish, this third question is concerned with how to develop the skill efficiently. The optimization problem that the agent needs to address in this context is one of optimal exploration. I defined this optimization problem concretely, pointed out how it differs from the optimization problem typically addressed in the reinforcement learning literature, formulated the problem as an MDP, and presented

an algorithm for solving it approximately. This algorithm produces an exploration behavior that is qualitatively different from existing exploration methods and that results in dramatic improvements in empirical learning performance.

This dissertation departs from existing literature by explicitly and concretely defining a set of targets for the discovery process. Although ultimately what is needed is a discovery process, I argue that the process of defining what makes a skill useful is necessary for developing effective processes that can reliably succeed in large, complex real-world problems. I believe that paying more attention to this conceptual problem will speed up progress in the field.

Future work can build on the ideas presented here in a number of ways, most importantly by extending the skill characterization in several directions, further developing the discovery and exploration algorithms proposed here, and introducing alternative algorithms.

One important direction for extending the skill characterization is to define similar skills in continuous environments. My conjecture is that many useful skills in continuous environments, for example, grasping, are useful for fundamentally the same reasons as the skills defined here in discrete environments.

A second direction is to refine the skill characterization to refer only to the parts of the state description that are relevant for the skill objective. For instance, rather than representing a particular doorway and a specific behavior for reaching that particular doorway, a skill may represent a generic description of doorways and a generic behavior that succeeds in reaching any specific doorway. Such a skill is truly reusable because it may be executed in a state that has not been experienced before, as long as the relevant part of the state matches the skill description.

A third direction for extending the skill characterization is to represent skills not in isolation but in a hierarchy that explicitly specifies the role they play in achieving the objectives of other skills. For example, reaching the entrance to a building is, in

many cases, necessary for reaching the entrance to a particular room in the building. The explicit hierarchical representation supports the development and use of learning and planning algorithms that can fully take advantage of behavioral units of different granularity.

BIBLIOGRAPHY

Amarel, S. (1968). On representations of problems of reasoning about actions. In Michie, D., editor, *Machine Intelligence 3*, volume 3, pages 131–171. Elsevier/North-Holland, Amsterdam, London, New York.

Barto, A. G. and Şimşek, Ö. (2005). Intrinsic motivation for reinforcement learning systems. In *Proceedings of the Thirteenth Yale Workshop on Adaptive and Learning Systems*.

Barto, A. G., Singh, S., and Chentanez, N. (2004). Intrinsically motivated learning of hierarchical collections of skills. In *Proceedings of the Third International Conference on Development and Learning*.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

Bonarini, A., Lazaric, A., Restelli, M., and Vitali, P. (2006). Self-development framework for reinforcement learning agents. In *Proceedings of the Fifth International Conference on Development and Learning*.

Bradke, S. J. and Duff, M. O. (1995). Reinforcement learning methods for continuous-time Markov decision problems. In Tesauro, G., Touretzky, D., and Lenn, T., editors, *Advances in Neural Information Systems*, volume 7, pages 393–400. The MIT press.

Brandes, U. (2001). A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23.

Burridge, R., Rizzi, A. A., and Koditschek, D. E. (1999). Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research*, 18:534–555.

Dawson, C. and Siklossy, L. (1977). The role of preprocessing in problem solving systems. In *Proc. of the 5th IJCAI*, pages 465–471, Cambridge, MA.

Dietterich, T. G. (2000). Hierarchical reinforcement learning with the MAXQ value function decomposition. *Journal of Artificial Intelligence Research*, 13:227–303.

Digney, B. (1998). Learning hierarchical control structure for multiple tasks and changing environments. In *From Animals to Animats 5: The Fifth Conference on the Simulation of Adaptive Behaviour*. MIT Press.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification*. Wiley, New York.

Duff, M. (2002). *Optimal learning: Computational procedures for Bayes-adaptive Markov Decision Processes*. PhD thesis, University of Massassachusetts Amherst.

Duff, M. (2003). Design for an optimal probe. In *Proceedings of the Twentieth International Conference on Machine Learning*.

Freeman, L. C. (1977). A set of measures of centrality based upon betweenness. *Sociometry*, 40:35–41.

Freeman, L. C. (1979). Centrality in social networks: Conceptual clarification. *Social Networks*, 1:215–239.

Hagen, L. and Kahng, A. B. (1992). New spectral methods for ratio cut partitioning and clustering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11:1074–1085.

Harel, D. (1987). Statecharts: A visual formulation for complex systems. *Science of Computer Programming*, 8(3):231–274.

Hengst, B. (2002). Discovering hierarchy in reinforcement learning with HEXQ. In Sammut, C. and Hoffmann, A. G., editors, *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 243–250. Morgan Kaufmann.

Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. The M.I.T. Press.

Iba, G. A. (1989). A heuristic approach to the discovery of macro-operators. *Machine Learning*, 3(4):285 – 317.

Jonsson, A. and Barto, A. G. (2005). A causal approach to hierarchical decomposition of factored MDPs. In *ICML*, pages 401–408.

Kaelbling, L. P., Littman, M. L., and Moore, A. W. (1996). Reinforcement learning: a survey. *Journal of Artificial Intelligence Research*, 4:237–285.

Kakade, S. and Dayan, P. (2001). Dopamine bonuses. In Leen, T. K., Dietterich, T. G., and Tresp, V., editors, *Advances in Neural Information Processing Systems*, volume 13, pages 131–137. MIT Press.

Kaplan, F. and Oudeyer, P.-Y. (2003). Motivational principles for visual know-how development. In Prince, C. G., Berthouze, L., Kozima, H., Bullock, D., Stojanov, G., and Balkenius, C., editors, *Proceedings of the Third International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*.

Kearns, M. and Singh, S. (1998). Near-Optimal reinforcement learning in polynomial time. In Shavlik, J. W., editor, *Proceedings of the Fifteenth International Conference on Machine Learning*, pages 260–268. Morgan Kaufmann.

Korf, R. E. (1985). Macro-operators: A weak method for learning. *Artificial Intelligence*, 26(1):35–77.

Mahadevan, S. (2005). Proto-value functions: Developmental reinforcement learning. In *Proceedings of the International Conference on Machine Learning*.

Mahadevan, S. and Maggioni, M. (2007). Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research*, 8:2169–2231.

Mannor, S., Menache, I., Hoze, A., and Klein, U. (2004). Dynamic abstraction in reinforcement learning via clustering. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 560–567. ACM Press.

Marthi, B., Kaelbling, L., and Lozano-Perez, T. (2007). Learning hierarchical structure in policies. *NIPS Workshop on Hierarchical Organization of Behavior*.

Mathew, V. (2008). Automated spatio-temporal abstraction in reinforcement learning. Master's thesis, Indian Institute of Technology Madras.

McGovern, A. (2002). *Autonomous discovery of temporal abstractions from interaction with an environment*. PhD thesis, University of Massachusetts, Amherst.

McGovern, A. and Barto, A. G. (2001). Automatic discovery of subgoals in reinforcement learning using diverse density. In Brodley, C. E. and Danyluk, A. P., editors, *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 361–368. Morgan Kaufmann.

McGovern, A., Sutton, R. S., and Fagg, A. H. (1997). Roles of macro-actions in accelerating reinforcement learning. In *Grace Hopper Celebration of Women in Computing*, pages 13–18.

Mehta, N., Ray, S., Tadepalli, P., and Dietterich, T. (2008). Automatic discovery and transfer of maxq hierarchies. In *International Conference on Machine Learning*.

Menache, I., Mannor, S., and Shimkin, N. (2002). Q-Cut - Dynamic discovery of sub-goals in reinforcement learning. In Elomaa, T., Mannila, H., and Toivonen, H., editors, *Proceedings of the Thirteenth European Conference on Machine Learning*, volume 2430 of *Lecture Notes in Computer Science*, pages 295–306. Springer.

Moore, A. and Atkeson, C. G. (1993). Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130.

Parr, R. (1998). *Hierarchical Control and Learning for Markov Decision Processes*. PhD thesis, Computer Science Division, University of California, Berkeley.

Parr, R. and Russell, S. (1998). Reinforcement learning with hierarchies of machines. In Jordan, M. I., Kearns, M. J., and Solla, S. A., editors, *Advances in Neural Information Processing Systems*, volume 10, pages 1043–1049. MIT Press.

Peng, J. and Williams, R. J. (1993). Efficient learning and planning within the dyna framework. *Adaptive Behavior*, 2:437–454.

Pickett, M. and Barto, A. G. (2002). PolicyBlocks: An algorithm for creating useful macro-actions in reinforcement learning. In Sammut, C. and Hoffmann, A. G., editors, *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 506–513. Morgan Kaufmann.

Precup, D. (2000). *Temporal abstraction in reinforcement learning.* PhD thesis, University of Massachusetts Amherst.

Puterman, M. L. (1994). *Markov Decision Processes.* Wiley.

Schmidhuber, J. (1991). A possibility for implementing curiosity and boredom in model-building neural controllers. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior.*

Schmidhuber, J. and Storck, J. (1993). Reinforcement driven information acquisition in nondeterministic environments. Technical report, Fakultat fur Informatik, Technische Universit at Munchen.

Shi, J. and Malik, J. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905.

Şimşek, Ö. and Barto, A. G. (2004). Using relative novelty to identify useful temporal abstractions in reinforcement learning. In *Proceedings of the Twenty-First International Conference on Machine Learning*, pages 751–758. ACM Press.

Şimşek, Ö. and Barto, A. G. (2006). An intrinsic reward mechanism for efficient exploration. In *Proceedings of the Twenty-Third International Conference on Machine Learning.*

Şimşek, Ö. and Barto, A. G. (2009). Skill characterization based on betweenness. In *Advances in Neural Information Processing Systems*, volume 21 (to appear).

Şimşek, Ö., Wolfe, A. P., and Barto, A. G. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Proceedings of the Twenty-Second International Conference on Machine Learning.*

Singh, S., Barto, A. G., and Chentanez, N. (2005). Intrinsically motivated reinforcement learning. In *Advances in Neural Information Processing Systems.*

Stolle, M. (2004). Automated discovery of options in reinforcement learning. Master's thesis, McGill University.

Stolle, M. and Precup, D. (2002). Learning options in reinforcement learning. In Koenig, S. and Holte, R. C., editors, *Abstraction, Reformulation and Approximation, 5th International Symposium, SARA 2002, Kananaskis, Alberta, Canada, August 2-4, 2002, Proceedings*, volume 2371 of *Lecture Notes in Computer Science*, pages 212–223. Springer.

Sutton, R. S. (1990). Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In Porter, B. W. and Mooney, R. J., editors, *Proceedings of the Seventh International Conference on Machine Learning*, pages 216–224. Morgan Kaufmann.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.

Sutton, R. S., Precup, D., and Singh, S. P. (1999). Between MDPs and Semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211.

Thrun, S. (1992). Efficient exploration in reinforcement learning. Technical Report CMU-CS-92-102, Carnegie-Mellon University.

Thrun, S. and Schwartz, A. (1995). Finding structure in reinforcement learning. In Tesauro, G., Touretzky, D. S., and Leen, T. K., editors, *Advances in Neural Information Processing Systems*, volume 7, pages 385–392. MIT Press.

Wasserman, S. and Faust, K. (1994). *Social Network Analysis*. Cambridge University Press, Cambridge U.K.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, Cambridge University.

Weng, J., McClelland, J., Pentland, A., Sporns, O., Stockman, I., Sur, M., and Thelen, E. (2001). Autonomous mental development by robots and animals. *Science*, 291:599–600.

Wu, Z. and Leahy, R. (1993). An optimal graph theoretic approach to data clustering: Theory and its application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15:1101–1113.