Automated Spatio-Temporal Abstraction in Reinforcement Learning

A THESIS

submitted by

VIMAL MATHEW

for the award of the degree

of

MASTER OF SCIENCE
(by Research)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, MADRAS.

May 2007

THESIS CERTIFICATE

This is to certify that the thesis titled **Automated Spatio-Temporal Abstraction in Reinforcement Learning**, submitted by **Vimal Mathew**, to the Indian Institute of Technology, Madras, for the award of the degree of **Master of Science (by Research)**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. B. Ravindran
Research Guide
Assistant Professor
Dept. of Computer Science and Engineering
IIT-Madras, 600 036

Place: Chennai

Date:

ACKNOWLEDGEMENTS

I would like to thank my research guide, Dr. B. Ravindran, for giving me the freedom and encouragement to pursue my research dreams, listening patiently to many harebrained ideas.

The friends I made during my stay at IIT have enriched my life. I would especially like to thank Sree Hari Krishnan and Shravan. The many hour-long discussions on all matters profound and profane at the Bermuda-quadrangle have broadened my perspective and clarified my thoughts. I would also like to thank Raj Gupta. A constant companion at the hostel mess, I have come to respect his willpower and determination to succeed.

I am indebted to my parents and my sister Vinitha for their understanding and encouragement when it was most required. They have stood by me through some tough decisions and given me the courage to move on.

The works of literature by Isaac Asimov have been a deep source of inspiration for me.

ABSTRACT

KEYWORDS: Reinforcement Learning ; Abstraction; Dynamical Systems ; Laplacian.

Reinforcement learning is a machine learning framework in which an agent manipulates its environment through a series of actions, receiving a scalar feedback signal or reward after each action. The agent learns to select actions so as to maximize the expected total reward.

The nature of the rewards and their distribution across the environment decides the task that the agent learns to perform. Consider a generic agent solving multiple tasks in a common environment. The effectiveness of such an agent increases with its ability to reuse past experience in new tasks. Identifying similar representations between tasks paves the way for a possible transfer of knowledge between such tasks. A sufficiently broad definition of similarity and an ability to effectively store and reuse previous experience can greatly increase the long-term performance of a multipurpose agent.

This thesis introduces an automated task-independent abstraction method that generalizes across tasks sharing the same state-space topology and similar dynamics. New tasks performed on this common environment are able to reuse skills acquired over previous tasks. Pruning techniques are identified for specific situations to identify relevant skills.

The interaction of the agent with the environment is modeled as a dynamical system. Identification of metastable regions in the agent dynamics leads to a partitioning of the state space. Transitions between metastable regions, which normally are events of low probability during a random-walk on the state space, are identified as useful skills.

TABLE OF CONTENTS

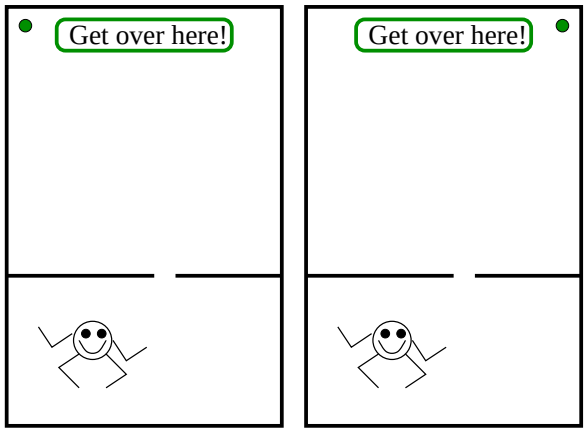
LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

The ability of a generic learning agent to look beyond the nature of an immediate task may provide a basis for improved generalization across tasks. An agent that is intrinsically motivated to build a broad competence in its environment, independent of the current task, is likely to improve performance over time. In this thesis we use ideas from dynamical systems to obtain a novel approach to task-independent state abstraction in Reinforcement Learning. Optimal strategies to transit between such abstract states are acquired to obtain a knowledge-base of useful skills. These skills are reused in new tasks and lead to a more competent learning agent.



(a) Reach top-left corner **(b)** Reach top-right corner

Figure 1.1: Abstraction and Task transfer

Consider a simple example (Figure 1.1) consisting of two rooms connected by a door. Assume that locations in this environment are described by discrete states as viewed by a learning agent. The agent can take actions *North*, *South*, *East*

and *West* to move around, with each action causing a change in the location of the agent. The change in location of the agent is mirrored in the change of state as viewed by the agent. The set of all states could be partitioned based on the room to which each state belongs, with each room corresponding to an abstract state. The environment can then be described, at a high level, in terms of two interconnected states and transitions between these states. A transition between the two rooms would correspond to a sequence of actions over the lower-level states, leading to a temporal abstraction. Therefore, a set of temporal abstractions can be defined in terms of transitions between abstract states. In the example given, the temporal abstractions correspond to *MoveToUpperRoom* and *MoveToLowerRoom*. Further, the aggregation of states as we move to a higher-level implies that the environment can be described in terms of fewer (abstract) states. Solving a task over this smaller state space has computational advantages. By repeating this process of spatio-temporal abstraction, a hierarchy of abstractions can be defined over the environment. This framework of using abstraction to simplify problem solving has been studied for some time [3, 16, 30].

Consider two different tasks on this common environment that correspond to moving either to the uppper-left corner (Figure 1.1a) or to the upper-right corner (Figure 1.1b). If both these tasks share the same abstractions as described above, their solutions can be described in terms of the subtask *MoveToUpperRoom* followed by a sequence of transitions to the appropriate corner. A solution to the subtask obtained while solving any one of the tasks can be directly reused to solve the other task. Therefore, similar abstractions can provide a basis for knowledge transfer between tasks.

1.1 Motivation

Curiosity drives human-beings to understand the environment around them. In the process of interacting with the environment and exploring it, often accomplishing mundane tasks such as driving to work or walking around in a building, we construct abstract models of the environment. These models are built independent of specific tasks, but mirror the environment. Such an internal representation of the world remains consistent across tasks that share the same environment and enables us to reuse skills across different tasks. The skills that we learn over time enable us to gain greater control over the environment in the long term.

To mirror this approach in the design of learning agents, there is a need to design algorithms that can automatically construct task-independent state abstractions and define skills in terms of these abstractions. Such skills will enable an agent to generalize across tasks and reuse experience more effectively.

Current approaches focus either on automated state abstraction or transferring knowledge across tasks. A few methods that have gained some success in combining state abstraction with knowledge transfer make restrictions on state representations or require domain knowledge. There is a need for a framework that is capable of both automated state abstraction and knowledge transfer across tasks, without making restrictive assumptions on state representations or requiring prior knowledge.

1.2 Objectives

The Reinforcement Learning framework is used to model a problem-solving learning agent able to interact with an environment. Tasks are specified in terms of Markov Decision Processes (MDPs) [44]. Given a collection of tasks that share the same

environment and transition dynamics but differ only in the nature of tasks

- Generate a state-abstraction consistent across all such tasks.
- Identify and acquire useful skills.
- Reuse relevant skills to solve new tasks.
- Automate the entire process without using prior domain knowledge.

1.3 Outline of Thesis

The first chapter motivates and defines objectives for the thesis. Chapter 2 introduces basic notation and background, relating the current work with existing literature to provide a context to the rest of the thesis. Some of our work on Deictic Option Schemas extending existing literature on skill transfer across tasks is also described, though not directly related to the main topic of the thesis.

Chapters 3 through 5 constitute the body of the thesis. Chapter 3 provides background to some of the theory used. It provides a brief introduction to dynamical systems and describes state abstraction in terms of a separation of time-scales. The description of a dynamical system is shown to emerge out of the time-scale of observation. A random-walk on a graph is described in terms of dynamical systems. The random-walk operator is shown to be related to graph Laplacians.

Chapter 4 defines a random-walk operator in a Reinforcement Learning problem, generalizes it across a global state space, and uses it to identify a partition on the state space. This state abstraction is generated independent of the nature of the task implied in the MDP. Some aspects of the state-abstraction method are explored using a few simple experiments.

Chapter 5 identifies temporal abstractions in terms of transitions between abstract states. Connections are drawn with the Intrinsically Motivated Reinforcement Learning framework and the Proto-value function framework.

The thesis concludes with chapter 6 with a discussion on some limitations, and ideas for further improvements.

CHAPTER 2

Background and Related work

This chapter gives a brief introduction of Reinforcement Learning (RL) along with its formulation as a Markov Decision Problem (MDP). This is followed by a look at existing literature, thereby giving a context to the rest of the thesis. A section on Deictic Option Schemas describes some of our work, not directly related to the main topic of the thesis, extending literature on skill transfer across tasks.

2.1 Background

2.1.1 Reinforcement Learning

A Reinforcement Learning (RL) [52] task involves an *agent* that interacts with a stochastic *environment* by taking *actions* and obtains a scalar *reward* signal. The environment responds to actions taken by the agent and presents a new situation to the agent which the agent represents internally as the *state* of the environment. The rewards observed during interaction depend on actions performed in a given state and are used to learn an optimal *policy*. An optimal policy maximizes the expected total reward starting from any *state* in the environment, and hence solves a sequential decision problem. A stochastic policy $\pi(s, a) \rightarrow [0, 1]$ determines the behavior of the agent in a given state. There are many approaches to define cumulative reward but here we restrict ourselves to the discounted reward formulation. For a state-action sequence $s_0, a_0, s_1, a_1, s_2, a_2 \ldots$ the cumulative discounted reward

is given by $R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \ldots$ with the *discount factor* γ lying between 0 and 1. The discount factor describes the preference of an agent for current rewards over future rewards.

With discounted rewards, the total reward for any state sequence is finite for finite step rewards.

2.1.2 Markov Decision Process

A Markov Decision Process (MDP) is a tuple $\langle S, A, \Psi, P, R \rangle$[1] where S is the set of states, A the set of actions, $\Psi \subseteq S \times A$ is the set of admissible state-action pairs, $P : \Psi \times S \to [0, 1]$ is the transition probability function with $P(s, a, s')$ being the probability of transition from state s to state s' under action a, and $R : \Psi \times S \to \mathbb{R}$ is the expected reward function, with $R(s, a, s')$ being the expected reward for the transition from s to s' under action a. A finite MDP has finite state and action sets. Further discussion on RL is restricted to finite MDPs and bounded rewards. The reward structure along with the goal state defined on an MDP identifies the task that the RL agent is expected to perform.

A *stochastic policy* $\pi : \Psi \to [0, 1]$ is the probability of picking an action a in a state s. For a policy π, a utility or *value function* can be associated with each state and computed in terms of the expected utility or values of state sequences that follow from that state under the policy. The value function satisfies the *Bellman equation*

$$V^{\pi}(s) = \sum_{a \in A(s)} \pi(s, a) \sum_{s' \in S} P(s, a, s') \left\{ R(s, a, s') + \gamma V^{\pi}(s') \right\}$$

[1]When there is no need to explicitly specify valid state-action pairs, we omit Ψ from the MDP.

where $0 \leq \gamma < 1$ is the discount factor, and $A(s) = \{a | \exists s' \in S, P(s, a, s') > 0\}$ is the set of feasible actions in state s

For the optimal policy π^*, the associated value function V^* is given by

$$V^*(s) = \max_{\pi} \sum_{a \in A(s)} \pi(s, a) \sum_{s' \in S} P(s, a, s') \left\{ R(s, a, s') + \gamma V^*(s') \right\}$$

so that the optimal policy is given by

$$\pi^*(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s' \in S} P(s, a, s') \left\{ R(s, a, s') + \gamma V^*(s') \right\}$$

A semi-Markov decision process (SMDP) is a generalization of an MDP in which actions can take variable amounts of time to complete. A finite discrete time SMDP can be written as a tuple $\langle S, A, \Psi, P, R \rangle$, where S, A and Ψ are the sets of states, actions and admissible state-action pairs; $P : \Psi \times S \times \mathbb{N} \rightarrow [0, 1]$ is the transition probability function with $P(s, a, s', N)$ being the probability of transition from state s to s' under action a in N time steps, and $R : \Psi \times \mathbb{N} \rightarrow \mathbb{R}$ is the expected reward function, with $R(s, a, N)$ being the expected reward for performing action a in state s and completing it in N time steps.

2.1.3 The Options framework

The *Options* framework [53] is one of many approaches to represent temporal abstractions within Reinforcement Learning. This framework will be used later on in the thesis to represent useful skills after they have been identified.

An *option* [53] in an MDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ is defined by the tuple $O = \langle \mathcal{I}, \pi, \beta \rangle$, where the initiation set $\mathcal{I} \subseteq S$ is the set of states in which the option

can be invoked, π is the policy to be followed while executing the option, and the termination function $\beta : S \rightarrow [0, 1]$ gives the probability of the option terminating in any given state.

2.2 Related work

This section provides an overview of current literature on abstractions directed at knowledge transfer. Section 2.3 describes some contributions extending the MDP homomorphism framework.

Automated state-abstraction techniques help reinforcement-learning algorithms scale to larger problems. Multiple RL tasks can transfer learning based on similarities in their state-abstractions. The capability of automated state-abstraction combined with knowledge transfer can lead to practical implementations of RL agents. We look at some previous work, with methods focused on either automated state-abstraction or knowledge transfer across tasks. A few techniques that were able to combine these two subtasks are also studied.

The UTree algorithm [38] automatically identifies state abstractions by using decision-tree learning methods. It has been extended to the *Options* framework in a hierarchical learning system [28]. The algorithm provides a compact representation for an action-value function on the state space. This relation to action-values and the associated policy makes this method task-specific.

Methods that base state abstraction on the value-function or its approximation bias themselves toward a particular task. The abstractions generated will be limited to tasks with similar goals.

MDP homomorphisms [45] provide a framework to model abstractions. An MDP

homomorphism is a transformation between two MDPs that preserves transition and reward structure to map equivalent states to each other. Therefore, the transformation between two MDPs preserves value functions over equivalent states. The strict requirement of exact homomorphisms was relaxed to define approximate MDP homomorphisms. Extensions to SMDPs [47] provide a way to model abstractions at multiple levels in a hierarchy. Given a set of MDPs related to each other through MDP homomorphisms, a solution to any one MDP can be transformed to obtain a solution for any other MDP in the set given the right transformation. Therefore identifying the right transformation [46] is sufficient to solve a related MDP. The similarities required in both transition and reward distributions over the state space limits the set of real-world tasks where such methods may be used.

The VISA algorithm [27] decomposes factored MDPs into hierarchies of options. It requires a DBN model of the factored MDP, using it to construct a causal graph that describes the relation between state variables. An analysis of state variable changes is used to induce hierarchies of temporal abstractions using the Options framework. The algorithm assumes *key* state variables to change at different time scales. Variables that change slowly correspond to higher levels of abstraction, and temporal abstractions are constructed to cause these changes. The constraints on state representations along with dependence on a DBN model are drawbacks of the algorithm.

McGovern [39] identifies *bottleneck states* in the state space. Such states separate strongly connected regions in the state space. Successful and unsuccessful trajectories are defined in a problem-specific manner such that bottleneck states lie only on successful trajectories, but never on unsuccessful trajectories. The concept of diverse-density from multiple-instance learning problems is used to identify

bottleneck regions. Task-specific trajectories are studied to identify transitions to bottleneck states. Such transitions are abstracted in terms of subgoals. The task-dependent definition of successful or unsuccessful trajectories reduces the degree of automation in state abstraction. The temporal abstractions that lead to a bottleneck are defined in a task-specific manner and may not easily generalize to other tasks.

Mannor et al. [37] suggests state abstraction using clustering methods based on the topology of the state space. A state-transition graph is constructed with states corresponding to nodes and valid transitions corresponding to edges between nodes. An objective function is used to cluster the nodes such that clusters are well-separated and of similar sizes. The well-separation criterion is similar to the McGovern [39] notion of *bottlenecks*. The tendency to search for equal sized clusters might create spurious results when the topology of the state space breaks down into regions of unequal size. As an example, a simple two-room grid world with a large room connected to a much smaller room by a door might not be partitioned at the door. State spaces where the well-separation criterion conflicts with the similar-size criterion can lead to un-intuitive partitions.

Şimşek et al. [50] construct localized transition matrices and analyze them using graph theoretic methods to obtain *access states*. Access states are similar in nature to *bottlenecks* and allow more efficient exploration of the state space. A local cut on the state space is approximated using *NCut* [49] to identify N partitions on the state space. Partitions with low *NCut* value are filtered using sampling methods to obtain access states. For multiply-connected regions of the state space, a single local cut may not partition the state space. Further, the nature of localized cuts ignores the structure of the global state space and cannot identify access states that

may be related to each other. Temporal abstractions are constructed that use these access states as sub-goals.

The methods indicated above can be broadly classified as :

- Methods that automatically partition the state space into abstract states [38, 50]. Such methods are either task-specific or make strong assumptions on the state space.

- Methods that identify useful regions within the state space to construct temporal abstractions [37, 39]. These regions identified may be task-independent but the temporal abstractions constructed are specific to the given task. An analysis of the re-usability of such temporal abstractions across different tasks is needed.

- Methods that provide a framework to transfer learning across tasks but need domain knowledge, and cannot be completely automated [27, 45].

There is a need to combine state abstraction, temporal abstraction and knowledge transfer into a single framework. An ability to automate the entire process can lead to the design of general purpose learning agents.

2.3 Deictic Option Schema

This section extends and reinterprets the MDP homomorphism framework [46, 47] to define deictic option schema.

Deictic representations [1, 2] are based on selective attention to relevant features. Objects in the environment are identified by their role in a given task. Such representations are also useful in modeling systems with limited sensory capabili-

ties working in complex environments [40]. As an example, Agre and Chapman [2] designed an agent *Pengi* to solve the arcade game Pengo using complex pointers such as *bee-attacking-me*, *ice-cube-next-to-me* etc. Deictic representations have also gained interest in the reinforcement learning community [20, 38, 58].

In this section we develop a hierarchical deictic RL framework based on relativized options [46], extending it to factored MDPs. We also show that some aspects of deixis can be modeled as finding structured homomorphic reductions of the problem. A Bayesian algorithm is presented to learn the correct configuration of pointers and the approach is validated on a simulated game domain inspired by Agre and Chapman [2].

The rest of this section is organized as follows. An introduction to some notation is followed by a brief summary of relativized options. Deictic option schema are introduced followed by a Bayesian algorithm to identify the right pointer configuration for a given sub-task. This approach is then related to an existing family of deictic algorithms. The section concludes with some related work and possible directions of research.

2.3.1 Notation

Consider a structured (finite) MDP $\langle S, A, \Psi, P, R\rangle$ as defined earlier with $R(s,a)$ being the expected reward for performing action a in state s and given by $R(s,a) = \sum_{s'} R(s,a,s')P(s,a,s')$. Consider MDPs where the state space S can be completely described by M features or variables, such that $S \subseteq \prod_{i=1}^{M} S_i$, where S_i is the set of permissible values for feature i. Thus any $s \in S$ is of the form $s = \langle s_1, \ldots, s_M\rangle$, where $s_i \in S_i$ for all i.

The transition probability function P is usually described by a family of two-slice

temporal Bayesian networks (2-TBNs), with one TBN for each action. A 2-TBN is a two layer directed acyclic graph whose nodes are $\{\mathbf{s_1}, \ldots, \mathbf{s_M}\}$ and $\{\mathbf{s'_1}, \ldots, \mathbf{s'_M}\}$. Here $\mathbf{s_i}$ denotes the random variable representing feature i at the present state and $\mathbf{s'_i}$ denotes the random variable representing feature i in the resulting state. Many classes of structured problems may be modeled by a 2-TBN in which each arc is restricted to go from a node in the first set to a node in the second. The state-transition probabilities can be factored as:

$$P(s, a, s') = \prod_{i=1}^{M} \text{Prob}(s'_i | \text{Parents}(\mathbf{s'_i}, a)),$$

where Parents($\mathbf{s'_i}, a$) denotes the parents of node $\mathbf{s'_i}$ in the 2-TBN corresponding to action a and each Prob(s'_i|Parents($\mathbf{s'_i}, a$)) is given by a conditional probability table (CPT) associated with node $\mathbf{s'_i}$. In computing the conditional probabilities it is implicitly assumed that the nodes in Parents($\mathbf{s'_i}$,a) are assigned values according to s.

Consider temporally extended actions represented using the Options framework. In general, the option policy can be a mapping from arbitrary sequences of state-action pairs (or histories) to action probabilities. We restrict attention to Markov sub-goal options in which the policies are functions of the current state alone, and an option terminates on reaching a set of pre-defined (sub)goal states. The states over which the option policy is defined is known as the *domain* of the option. In such cases the option policy can be defined in a sub-MDP, known as the option MDP, consisting of the states in the domain of the option.

2.3.2 Relativized Options

A relativized option combines MDP homomorphisms [47] with the options framework to compactly represent a family of related options. An MDP homomorphism is a transformation from an MDP $\mathcal{M}$ to a reduced model $\mathcal{M}'$ such that a solution to $\mathcal{M}'$ yields a solution to $\mathcal{M}$. Notationally, an MDP homomorphism, h, is defined as a surjection from Ψ to Ψ', given by a tuple of surjections $\langle f, \{g_s | s \in S\}\rangle$, with $h((s,a)) = (f(s), g_s(a))$, where $f : S \to S'$ and $g_s : A_s \to A'_{f(s)}$ for $s \in S$. $\mathcal{M}'$ is called the *homomorphic image* of $\mathcal{M}$ under h. An optimal policy in $\mathcal{M}'$ when lifted to $\mathcal{M}$ yields an optimal policy in $\mathcal{M}$.

In a relativized option, the policy for achieving the option's sub-goal is defined in the image of a partial homomorphism defined only over a subset of S. This image is called the option MDP. When the option is invoked, the current state is projected onto the option MDP, $\mathcal{M}_O = \langle S_O, A_O, \Psi_O, P_O, R_O\rangle$, and the policy action is lifted to the original MDP based on the states in which the option is invoked. A relativized option is defined as follows:

Definition: A *relativized option* of an MDP $\mathcal{M} = \langle S, A, \Psi, P, R\rangle$ is the tuple $O = \langle h, \mathcal{M}_O, \mathcal{I}, \beta\rangle$, where $\mathcal{I} \subseteq S$ is the initiation set, $\beta : S_O \to [0,1]$ is the termination function and $h = \langle f, \{g_s | s \in S\}\rangle$ is a partial homomorphism from the MDP $\langle S, A, \Psi, P, R'\rangle$ to the option MDP $\mathcal{M}_O$ with R' chosen based on the sub-task. In other words, the option MDP $\mathcal{M}_O$ is a partial homomorphic image of an MDP with the same states, actions and transition dynamics as $\mathcal{M}$ but with a reward function chosen based on the option's sub-task. The homomorphism conditions hold only for states in the domain of the option O. The option policy $\pi : \Psi_O \to [0,1]$ is obtained by solving $\mathcal{M}_O$ by treating it as an episodic task. When lifted to $\mathcal{M}$, π is suitably transformed into policy fragments over Ψ.

A relativized option can be viewed as an *option schema* where a template of an option policy is specified using a parameterized representation of a family of sub-problems.

When the option is invoked, a particular instantiation of the schema is chosen by binding to the appropriate resources. Given a set of possible bindings, [46] presented a Bayesian approach to choosing the right binding to apply in a given context based on experience gathered in solving the task. It assumed that the set of bindings were given by a family of transformations, $\mathcal{H}$, applied to Ψ and maintained a heuristic weight vector, $w_n(h, \overline{s})$, which is a measure of the likelihood of transformation $h \in \mathcal{H}$ being the right transformation in the context represented by $\overline{s}$.[2] The weight vectors are updated using:

$$w_n(h, \overline{s}) = \frac{\overline{P_O}((f(s), g_s(a), f(s'))w_{n-1}(h, \overline{s})}{\mathcal{K}} \tag{2.1}$$

where $\overline{P_O}(s, a, s') = \max(\nu, P_O(s, a, s'))$, and

$$\mathcal{K} = \sum_{h' \in \mathcal{H}} \overline{P_O}((f'(s), g'_s(a), f'(s'))w_{n-1}(h', \overline{s})$$

is a normalizing factor. Since the projected transition probability is lower bounded by ν, this approach works even when the homomorphic image is not exact. In this chapter we extend earlier work [46] to cases where the bindings are specified by deictic pointers.

[2]Frequently, $\overline{s}$ is some simple function of the state space, like a projection onto a few features.

2.3.3 Deictic Option Schema

The set of candidate transformations, $\mathcal{H}$, can be specified by a set of deictic pointers together with their possible configurations. The agent learns to place the pointers in specific configurations to effect the correct bindings to the schema. We call such option schema together with the set of pointers a *deictic option schema*. Formally a deictic option schema is defined as follows:

Definition: A *deictic option schema* of a factored MDP $\mathcal{M} = \langle S, A, \Psi, P, R \rangle$ is the tuple $\langle \mathcal{D}, O \rangle$, where $O = \langle h, \mathcal{M}_O, \mathcal{I}, \beta \rangle$, is a relativized option in $\mathcal{M}$, and $\mathcal{D} = \{D_1, D_2, \cdots, D_K\}$ is the set of admissible configurations of the deictic pointers, with K being the number of deictic pointers available. For all i, $D_i \subseteq 2^{\{1,\cdots,M\}}$ is the collection of all possible subsets of indices of the features that pointer i can project onto the schema, where M is the number of features used to describe S.

The set D_i indicates the set of objects that pointer i can point to in the environment. For example, in a blocks world domain this is the set of all blocks. In our example in the next section, it is the set of all possible adversaries.

Once a deictic option schema is invoked the decision making proceeds in two alternating phases. In the first phase the agent picks a suitable pointer configuration, using a heuristic weight function similar to the one given in Section 3. In the second phase, the agent picks an action, based on the perceptual information obtained from the pointers, using a Q-function. This calling semantics is similar to that followed by Whitehead and Ballard [58]

Each member of $\mathcal{H}$ has a state transformation of the form $\prod_{i=1}^{K} \rho_{J_i}$, where $J_i \in D_i$ for all i and ρ_J, is the projection of S onto the subset of features indexed by J. If h is known a priori then the pointer configurations can be chosen appropriately while

learning. In the absence of prior knowledge the Bayesian algorithm developed earlier [46] can be used to determine the correct bindings to the schema from among the possible pointer configurations. But, the algorithm is not entirely suitable for deictic option schema for the following reason.

The algorithm assumes that the candidate transformations are not structured and maintains a monolithic weight vector, $w_n(\cdot,\cdot)$. In the case of deictic option schema the transformations are structured and it is advantageous to maintain a "factored" weight vector, $w_n(\cdot,\cdot) = \langle w_n^1(\cdot,\cdot), w_n^2(\cdot,\cdot), \cdots \rangle$. Ideally each component of the weight vector should be the likelihood of the corresponding pointer being in the right configuration. But usually there is a certain degree of dependence among the pointers and the correct configuration of some pointers might depend on the configuration of other pointers.

Therefore, three cases need to be considered. Assume that there are only two pointers, i and j, for the following discussion, but the concepts generalize to arbitrary number of pointers.

1. *Independent pointers:* For every $J \in D_i$, ρ_J satisfies the homomorphism condition on transition probabilities. Then, the right assignment for pointer i is independent of the other pointers and there is one component of the weight vector corresponding to pointer i and the updates for this components depends only on the features indexed by some $J \in D_i$.

2. *Mutually dependent pointers:* For each $J \in D_i$ and $J' \in D_j$, $\rho_J \times \rho_{J'}$ satisfies the homomorphism conditions. But ρ_J and $\rho_{J'}$ do not satisfy the homomorphism conditions for some $J \in D_i$ and $J' \in D_j$. Thus, they cannot be treated separately and the composite projections given by their cross-products has to

be considered. There is one component of the weight vector that corresponds to this cross-product projection. The update for this component will depend on the features indexed by some $J \in D_i$ and $J' \in D_j$.

3. *Dependent pointer:* For each $J \in D_i$ and $J' \in D_j$, $\rho_J \times \rho_{J'}$ satisfies the homomorphism conditions, as does ρ_J. But $\rho_{J'}$ does not satisfy the homomorphism conditions for at least some value of $J' \in D_j$. This means pointer i is an independent pointer, while j is a dependent one. There is a separate component of the weight vector that corresponds to pointer j, but whose update depends on the features indexed by some $J \in D_i$ and $J' \in D_j$.

The weight vector is chosen such that there is one component for each independent pointer, one for each dependent pointer and one for each set of mutually dependent pointers. Let the resulting number of components be L. The weights are indicative of the probability of h being the correct transformation to apply in sub-problem $\overline{s}$. The weights are updated to take observed transitions into account in a manner similar to the recursive Bayes rule. The update rule used is a modified version of [46]. Each component l of the weight vector is updated independent of other components:

$$w_n^l(h^i, \overline{s}) = \frac{\overline{P_O^l}((f^i(s), g_s^i(a), f^i(s')) \cdot w_{n-1}^l(h^i, \overline{s})}{\mathcal{K}} \tag{2.2}$$

where $\overline{P_O^l}(s, a, s') = \max\left(\nu, P_O^l(s, a, s')\right)$, $\overline{s}$ is again a function of s that captures the features of the states necessary to distinguish the particular sub-problem under consideration, and $\mathcal{K} = \sum_{h'^i \in \mathcal{H}} \overline{P_O^l}(f'^i(s), g_s'^i(a), f'^i(s'))\ w_{n-1}(h'^i, \overline{s})$ is the normalizing factor. $P_O^l(s, a, s')$ is a "projection" of $P_O(s, a, s')$ computed as follows. Let J be the set of features that is required in the computation of $w_n^l(h^i, \overline{s})$. This is determined as

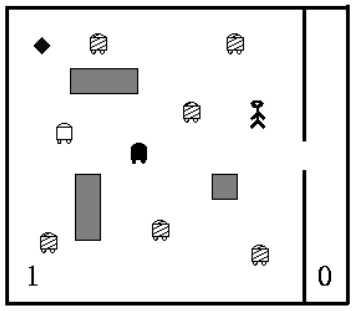


Figure 2.1: A game domain with interacting adversaries and stochastic actions. The task is to collect the black diamond. The adversaries are of three types—benign (shaded), retriever (white) and delayers (black). See text for more explanation.

described above for the various cases. Then $P_O^l(s, a, s') = \prod_{j \in J} \text{Prob}(s'_j | \text{Parents}(s'_j, a))$.

2.3.4 Experimental Illustration in a Game Environment

We now apply a deictic option schema to learning in a modified version of the game environment introduced in [46]. The layout of the game is shown in Figure 2.1. The environment has the usual stochastic grid-world dynamics. There is just one room in the world and the goal of the agent is to collect the diamond in the room and exit it. The agent collects a diamond by occupying the same square as the diamond. A Boolean variable *have* indicates possession of the diamond.

The room also has 8 autonomous adversaries. The adversaries may be of three types—benign, delayer or retriever. If the agent happens to occupy the same square as the delayer it is *captured* and is prevented from moving for a random number of time steps determined by a geometric distribution with parameter *hold*. When not occupying the same square, the delayer pursues the agent with probability *chase*. The benign robots execute random walks in the room and act as mobile obstacles. The retriever behaves like the benign adversary till the agent picks up the diamond.

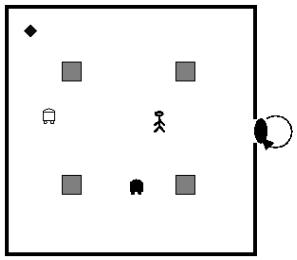


Figure 2.2: The option MDP corresponding to the sub-task *get-object-and-leave-room* for the domain in Figure 2.1. There is just one delayer and one retriever in this image MDP.

Once the agent picks up the diamond, the retriever's behavior switches to that of the delayer. The main difference is that once the retriever occupies the same square as the agent, the diamond is returned to the original position and the retriever reverts to benign behavior. The retriever returns to benign behavior if the agent is also "captured" by the delayer. None of the adversaries leave the room, and thus the agent can "escape" from the room by exiting to the corridor. The agent is not aware of the types of the individual adversaries.

The option MDP (Figure 2.2) is a symmetrical room with just two adversaries—a delayer and a retriever with fixed *chase* and *hold* parameters. The features describing the state space of the option MDP consists of the x and y coordinates relative to the room of the agent and of the adversaries and a boolean variable indicating possession of the diamond. The room in the world does not match the option MDP exactly and no adversary in the world has the same chase and hold parameters as the adversaries here.

The deictic agent has access to 2 pointers: a *delayer* pointer that projects one of the adversaries onto the delayer in the image MDP and a *retriever* pointer that projects one of the adversaries onto the retriever in the image MDP. The *delayer* pointer is an independent pointer and the *retriever* pointer is dependent on the

delayer pointer. The sets $D_{delayer}$ and $D_{retriever}$ are given by the 8 pairs of features describing the adversary coordinates.

In addition to the pointers the agent also has access to some background information, such as its own location (which can be formalized as a self pointer) and whether it has the diamond or not. Note that since the option MDP is an approximate homomorphic image, the homomorphism conditions are not strictly met by any of the projections. Therefore, in computing the weight updates, the influence of the features not used in the construction of the image MDP are ignored by marginalizing over them.

Experimental Results

Experimental trials conducted with a deictic agent as the learning agent are obtained for the option MDP (Figure 2.2). For comparison, trials are also conducted using a relativized agent (monolithic agent) as the learning agent that employs the same option MDP but chooses from a set $\mathcal{H}$ of 64 monolithic transformations, formed by the cross product of the 8 configurations of the deictic pointers. Both agents employ hierarchical SMDP Q-learning [12], with the learning rates for the option and the main task set to 0.1. The agents are both trained initially in the option MDP to acquire an approximate initial option policy that achieves the goal some percentage of the trials, but is not optimal. Both agents use ϵ greedy exploration. The results reported are averaged over 10 independent runs.

On learning trials both agents perform similarly (Figure 2.3), but the monolithic agent has a marginally better initial performance. To understand this we look at the rates at which the transformation weights converge (Figures 2.4, 2.5, and 2.6 are for a single typical run). Figure 2.5 shows that typically the deictic agent identifies

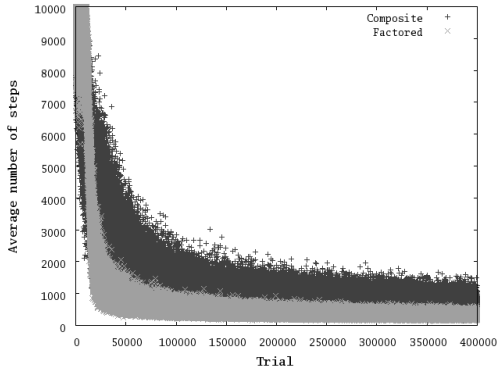


Figure 2.3: Average number of steps per episode taken by both agents for solving the task shown in Figure 2.1.

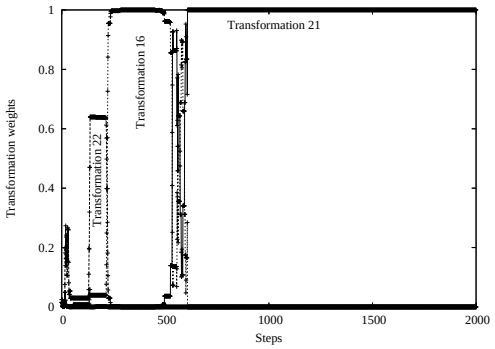


Figure 2.4: Typical evolution of a subset of weights of the monolithic agent on the task shown in Figure 2.1.

the *delayer* quite rapidly. In fact it takes only an average of 52 update steps to identify the delayer, over the 10 independent runs. The monolithic agent takes much longer to identify the right transformation (number 21 in our encoding), as seen from Figure 2.4. On an average it takes around 2007 update steps. As Figure 2.6 shows, identifying the *retriever* is harder and the deictic agent takes about 3050 update steps on an average.

This result is not surprising, since the correct position for the *retriever* depends on position of the *delayer* pointer. Therefore, while the *delayer* is being learned, the weights for the *retriever* receive inconsistent updates and it takes a while for

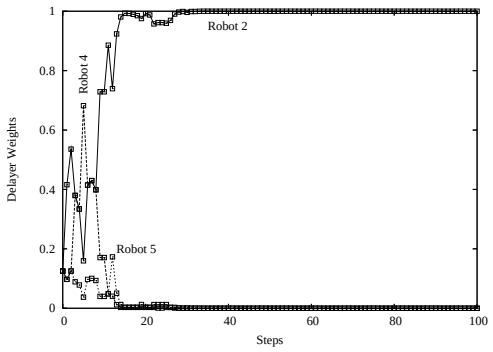


Figure 2.5: Typical evolution of a subset of the *delayer* weights of the deictic agent on the task shown in Figure 2.1.

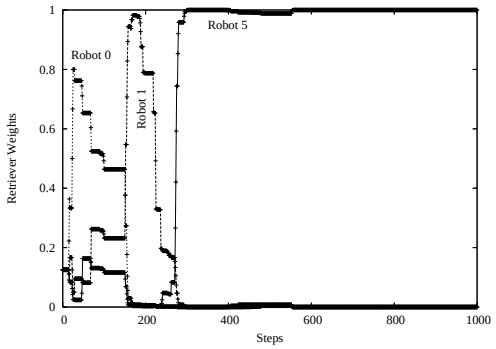


Figure 2.6: Typical evolution of a subset of the *retriever* weights on the task shown in Figure 2.1.

the weights to get back on track. For much of the time we are interested in only identifying the *delayer* correctly and hence the deictic agent seems to have lower variability in its performance. Overall a single run with the composite agent takes around 14 hours of CPU time on a Pentium IV, 3.4 GHz machine, while a single run of the deictic agent takes around 4 hours. Further, the monolithic agent considers all possible combinations of pointer configurations simultaneously. Therefore, while it takes fewer update steps to converge to the right weights, the deictic agent makes far fewer number of weight updates: 130,000 vs. 85,000 on an average.

Comment

The algorithm used above updates the weights for all the transformations after each transition in the world. This is possible since the transformations are assumed to be mathematical operations and the agent could use different transformations to project the same transition onto to the option MDP. But deictic pointers are often implemented as physical sensors. In such cases, this is equivalent to sensing every adversary in the world before making a move and then sensing them after making the move, to gather the data required for the updates. Since the weights converge fairly rapidly, compared to the convergence of the policy, the time the agent spends "looking around" would be a fraction of the total learning time.

2.3.5 Perceptual Aliasing and Consistent Representations

The power of deixis arises from its ability to treat many perceptually distinct states in the same fashion, but it is also the chief difficulty in employing deictic representations. This phenomenon is known as *perceptual aliasing* [58]. approach to overcome perceptual aliasing is a class of methods known as *Consistent Representation Methods*. These methods split the decision making into two phases: in the perceptual phase the agent looks around the environment to find a consistent representation of the underlying state. A consistent representation [58] of a state is one such that all states that map to the representation have the same optimal action-value function. In the overt phase the agent picks an action to apply to the environment based on the current sensory input. Learning takes place in both phases. The *Lion* algorithm [58] is an example of a consistent representation algorithm. Here Q-learning is used in the overt phase and a simple learning rule based on one step error information is used to train the sensory phase. If the one step error in the Q update rule for a par-

ticular configuration is negative then that representation is considered perceptually aliased and is ignored in the future. This simple rule limits the applicability of this algorithm to deterministic settings alone.

If the representation used is a homomorphic image then it is a consistent representation, as mentioned in [47]. By restricting the definition of deictic option schema to employ partial homomorphic images as option MDPs, it is guaranteed that a consistent representation is always employed. In the absence of knowledge of the option homomorphism, finding the right transformation to employ constitutes the search for a consistent representation and we employ Bayesian learning in this phase. As with the Lion algorithm, a form of Q-learning is used in the overt phase.

2.3.6 Deixis and Reinforcement Learning

Deixis originates from the Greek word *deiknynai* which means to show or to point out. It is employed by linguists to denote the pointing function of certain words, like *here* and *that*, whose meaning could change depending on the context. Deixis was introduced to the AI community by Agre. As mentioned earlier, [2] used deictic representations to design an agent, Pengi, that plays the arcade game Pengo. Pengi was designed to play the game from the view point of a human player and hence used visuals from a computer screen as input.

Whitehead and Ballard [58] were the first to use deictic representations in a RL system, with their *Lion* algorithm. Unfortunately, the method the Lion algorithm employs to determine consistency works only in deterministic environments. McCallum [38] takes a more direct approach to overcoming perceptual aliasing. He employs deixis to solve a car driving task and models the problem as a partially observable MDP. He uses a tree structure, known as U-trees, for representing "states"

and identifies the necessary distinctions so that the resulting representation is consistent. But his approach is not divided into explicit perceptual and overt phases. There has not been much work on using hierarchical RL and deixis. The only work we are aware of is by Minut and Mahadevan [40]. They develop a selective attention system that searches for a particular object in a room. It operates by identifying the most salient object in the agent's visual field and shifting its visual field to center and focus on that object. They employ an option to identify the most salient object in the current visual field. Though they do not state it thus, this is a "deictic" option, whose effect depends on the current visual field.

A systematic study on using deictic representations with RL was reported by Finney et al. [20]. They employ a straightforward deictic representation with two pointers on a blocks world task. They use the G-algorithm to represent past information as a tree. They report that their approach does not work well for a variety of reasons. First the tree grows very large rapidly. The deictic commands are defined with respect to the two focus pointers. When long sequences of actions are required with a small number of pointers, it is easy to lose focus. While they try to address this by redesigning the pointers, they do not have much success. One way to alleviate this problem is by adopting a hierarchical approach as we do in this work. If the number of pointers required by each deictic level to maintain focus is not large, we can avoid some of the problems encountered by Finney et al. [20].

2.3.7 Discussion

While deixis is a powerful paradigm ideally suited for situations that are mainly reactive, it is difficult to employ a purely deictic system to solve complex tasks that require long-range planning. Our hierarchical deictic framework allows us to

employ deictic representations in lower levels of the problem to leverage their power and generalization capacities, while at the higher levels we retain global context information in a non-deictic fashion. Mixing such representations allows us to exploit the best of both worlds and to solve tasks that require maintaining long term focus. It is our belief that there is no pure deictic system in nature. While it has been established that humans employ deixis in a variety of settings, we certainly maintain some higher level context information. While gathering ingredients for making tea, we might be using deictic pointers for accessing various containers [32], but we also are continuously aware of the fact that we are making tea.

The Bayesian approach we outlined requires that we have a complete model of the option MDP, which we assumed was available apriori. To learn this would require a lot more experience with the option MDP than we would need to learn a reasonable policy. Currently we are experimenting with learning a partial model of the option MDP, such that it is sufficient to identify the correct configuration of the deictic pointers.

The various approaches to learning with deictic pointers [20, 58] usually employ simple physical locators.Agre [1] uses complex pointers, but hand codes the policy for maintaining the focus of these pointers. For example, a set of rules are used to determine which is the *bee-attacking-me* and the pointer is moved suitably. Our approach falls somewhere in between. We start by defining a set of simple pointers. As learning progresses the agent learns to assign these pointers consistently such that some of them take on complex roles. We can then assign semantic labels to these pointers such as *robot-chasing-me*.

It should be noted that a homomorphic image implies a consistent representation but the reverse is not true. The notion of a consistent representation is a more

general concept than a homomorphic image and corresponds to optimal-action value equivalence discussed in Givan et al. [21]. By restricting ourselves to homomorphic images we are limiting the class of deictic pointers that we can model. Further work is needed to extend our framework to include richer classes of deictic pointers and to employ memory based methods. Nevertheless we have taken the first steps in accommodating deictic representations in a hierarchical decision theoretic framework.

The Deictic Option Schema framework assumes a set of transformations available to transfer previously available policies between tasks. A deictic representation of the domain along with sufficient domain knowledge to enumerate possible transformations may not always be available to a learning agent. The existence of RL tasks that differ only in their bindings of deictic pointers may be relatively rare in real world situations, even when approximations are involved.

There is a need for a framework capable of transferring knowledge over a broader class of tasks without requiring prior domain knowledge.

CHAPTER 3

Dynamical systems and Abstraction

This chapter describes the Laplacian operator on both continuous and discrete spaces. An introduction to Dynamical Systems is followed by a look at some of the reasons for simple abstractions in dynamical systems. The graph Laplacian is shown to be closely connected to a random walk on a graph. This random walk can be described as a dynamical system and the theory of separation of time-scales in simple dynamical systems provides a basis for abstraction in random walks. The random-walk related to an RL task will be described in the next chapter and it will be used to arrive at a state abstraction.

3.1 Manifolds and the Laplacian operator

This section defines the Laplace-Beltrami operator on smooth manifolds and describes some properties of its spectrum. The following definitions are based on Berger [8], Zorich [61, 62].

A *topological space* (X, τ) consists of a set X and a system τ of subsets of X (called *open sets in* X) if

1. $\phi \in \tau; X \in \tau$
2. $\left(A = (\alpha; \tau_\alpha \in \tau)\right) \Longrightarrow \bigcup_{\alpha \in A} \tau_\alpha \in \tau$
3. $\left(\tau_i \in \tau; i = 1, \ldots, n\right) \Longrightarrow \bigcap_{i=1}^{n} \tau_i \in \tau$

Therefore τ contains the empty set and the whole set X, the union of an uncountably infinite number of sets of τ is a set of τ, and intersection of any finite number of set of τ is a set of τ.

A *base* of the topological space (X, τ) is a family $\mathfrak{B}$ of open subsets of X such that every open set $G \in \tau$ is the union of some collection of elements of the family $\mathfrak{B}$.

A *neighborhood* of a point of a topological space (X, τ) is an open set containing the point.

A topological space is *Hausdorff* if any two distinct points in the space have non-intersecting neighborhoods.

A Hausdorff topological space whose topology has a countable base is called a *n-dimensional manifold* if each of its points has a neighborhood U homomorphic either to all of $\mathbb{R}^n$ or to the half-space $H^n = \{x \in \mathbb{R}^n | x^1 \leq 0\}$. The homomorphism is realized by a *local chart* $\phi : \mathbb{R}^n \to U \subset M$ (or $\phi : H^n \to U \subset M$, $\mathbb{R}^n$) (or H^n) is called the *parameter domain*, and U is called the *range* of the chart on the manifold M.

The set of functions $f : E \to \mathbb{R}$ having continuous derivatives up to order n inclusive will be denoted $C^{(n)}(E, \mathbb{R})$, or by the simpler $C^{(n)}$ when the domain and range are known. A set of charts whose ranges taken together cover the entire manifold is called an *atlas* of the manifold. An atlas of a manifold is *smooth* (of class $C^{(k)}$ or *analytic*) if all the coordinate-changing functions for the atlas are smooth mappings (diffeomorphisms) of the corresponding smoothness class. Two atlases of a given smoothness are *equivalent* if their union is an atlas of this smoothness.

A *smooth manifold* (*of class* $C^{(k)}$ or *analytic*) is a manifold M with an equivalence class of atlases of the given smoothness. For a smooth manifold with an atlas of

at least $C^{(1)}$ smoothness, every point m can be associated with a tangent space, denoted by T_mM, and its elements are called the *tangent vectors*. The tangent vector can be interpreted as an instantaneous velocity of motion. The tangent space is a vector space that has the same dimension as the manifold.

A *Riemannian metric* g on a smooth manifold associates each point m on the manifold with an inner product on the tangent space T_mM. A manifold with an associated Riemannian metric is called a *Riemannian manifold* (M, g).

A Riemannian metric g allows the definition of an intrinsic second derivative. For every smooth function

$$f : M \to \mathbb{R}$$

the Hessian $Hessf$ is made up of the derivatives of f. Taking the trace of the Hessian with respect to the metric g gives the Laplace-Beltrami operator

$$\Delta f = -div(\nabla f) = -trace_g Hessf$$

The Laplace-Beltrami operator Δ is a generalization of the Euclidean representation of the Laplace operator to a Riemannian manifold. Further references to the Laplacian on a manifold correspond to the Laplace-Beltrami operator.

Along geodesics, the second derivative coincides with ordinary numerical second derivative. By definition of the trace with respect to g, the Laplacian at $m \in M$ can be written as

$$\Delta f(m) = -\sum_{i=1}^{d} \frac{d^2}{dt^2} f(\gamma_i(t))\Big|_{t=0}$$

where the γ_i are geodesics through m whose velocities at m form an orthonormal basis of T_mM.

The eigen-value problem for the Laplacian on a compact manifold can be written as

$$\Delta f = \lambda f$$

where $f = \phi$ is an *eigenfunction* and the number λ is the corresponding eigenvalue.

The set of all eigenvalues of Δ is called the *spectrum* of Δ

$$Spec(M) = \lambda_k = \{0 \leq \lambda_1 \leq \lambda_2 \leq \ldots\}$$

The eigenvalue $\lambda_0 = 0$ because the Riemannian manifold has no boundary, and corresponds to a constant eigenfunction. If the manifold is connected, the multiplicity of λ_0 is exactly one. In general, the multiplicity of the zero eigenvalue corresponds to the number of connected components of the manifold.

The Laplacian is a self-adjoint operator on the manifold, and its eigenfunctions form a complete set of real-valued orthonormal basis functions on $L^2(M)$. This also extends for the case of a Laplacian M with a boundary δM. The *Dirichlet boundary condition* corresponds to functions vanishing on the boundary δM. Fourier series theory, which corresponds to the domain being an interval in the line $\mathbb{R}$, extends to functions on manifolds with Dirichlet boundaries conditions. Therefore, any function vanishing on δM can be written as

$$f = \sum_{i=1}^{\infty} c_i \phi_i$$

where

$$c_i = \int_M f(m)\phi_i(m)dm$$

for each i $= 1, 2, \ldots$. If the manifold is considered as a vibrating membrane, the

eigenvalues λ_i correspond to the frequencies of vibration, also called *harmonics*, the lowest being the *fundamental tone* [8].

A weighted manifold is denoted by (M, μ) where M is a Riemannian manifold and μ is a measure on M with smooth positive density with respect to the Riemannian measure μ_0, that is $d\mu = h^2 d\mu_0$ where $h \in C^\infty(M), h > 0$. The weighted Laplace operator Δ_μ [23] is a natural generalization of the Laplace-Beltrami operator to weighted manifolds and is defined by

$$\Delta_\mu = -\frac{1}{h^2} div(h^2 \nabla)$$

3.2 Graph-Laplacians

For a finite number of points the manifold degenerates into a weighted graph $G = (V, E, W)$. Each point is represented by a node or vertex $v \in V$. Edges indicate connections between these nodes. The weight $W : E \to \mathbb{R}$ associates a weight with each edge. Given a graph with $n = |V|$ nodes, the edge weights W can be represented as a matrix with entry W_{ij} representing the edge weight between vertices i and j. If no edge exists between two vertices i and j, we set $W_{ij} = 0$. The diagonal matrix D is defined to have diagonal entries $D_{ii} = \sum_{j=1}^{n} W_{ij}$. For a binary weight matrix W, D represents the out-degree of the vertex and is called the valence matrix.

Table 3.1: Some popular graph Laplacians

Laplacian	Definition
Random-walk	$\Delta^{(rw)} = (I - D^{-1}W)$
Combinatorial	$\Delta^{(cm)} = (D - W)$
Normalized	$\Delta^{(norm)} = (I - D^{-1/2}WD^{-1/2})$

Graph Laplacians have gained popularity and have been used in semi-supervised

learning [7, 60], spectral clustering [18] and dimensionality reduction [6, 14]. The proto-value function framework [36] within RL uses the eigenvectors of the graph Laplacian for value-function approximation. Table 3.1 indicates some of the more commonly used Laplacians. The usage of graph Laplacians has often been justified by their relation to the Laplace-Beltrami operator. Hein et al. [26] provides convergence results on the graph Laplacians. The graph Laplacians converge to the Laplace-Beltrami operator when a uniform measure is defined on the graph. For a graph with a non-uniform measure the random-walk Laplacian $\Delta^{(rw)}$ alone converges to the weighted Laplace-Beltrami operator.

From the definition of the random-walk Laplacian $\Delta^{(rw)} = I - T$, where $T = D^{-1}W$ is a random-walk operator, it is clear that it shares the same eigenvectors as the random-walk operator T, and the eigenvalues are related by $\lambda_\Delta = 1 - \lambda_T$. The convergence properties of $\Delta^{(rw)}$ indicate its suitability for use for both uniform and non-uniform measures on the graph.

3.3 Dynamical systems

A dynamical system is a mathematical formalization defining the time-dependence of a point's position in some state space. The state of a system at any given time can be represented by a single point in an abstract space called a *state space* or *phase space* $\mathcal{M}$. The system evolves with time, modeled in terms of the change in state of a *representative point*. The evolution of such points constitutes the dynamics of the system and follows the *evolution rule* f^t. Though the theory of dynamical systems includes continuous-time evolution, we restrict ourselves to discrete-time evolution.

Assume the state space to be a d-dimensional manifold $\mathcal{M}$. For a deterministic

system, the evolution rule $f^t : \mathcal{M} \to \mathcal{M}$ tells the location of a point $x \in \mathcal{M}$ after a time interval t. The pair $(\mathcal{M}, f)$ constitutes a dynamical system [15].

Given an initial point x_0, the evolution rule generates a sequence of points $x(t) = f^t(x_0)$, the *trajectory* through the point $x_0 = x(0)$. The subset of points from $\mathcal{M}$ that belongs to the trajectory of a point x_0 is called the *orbit* of x_0. Trajectories can be broadly classified as

Stationary:	$f^t(x) = x$	For all t
Periodic:	$f^t(x) = f^{t+T_p}(x)$	For a minimum time period T_p
Aperiodic:	$f^t(x) \neq f^{t'}(x)$	For all $t \neq t'$

A connected subset of the state space $\mathcal{M}$ that maps onto itself on forward evolution is called an *attractor*. Each attractor in the state space has its own *basin of attraction*, the set of points that fall into the attractor on forward evolution. An attractor can be a fixed point, a periodic orbit, or an aperiodic orbit.

In many systems the evolution of the system with time involves some uncertainty. Such a *stochastic system* can be modeled in terms of the time-evolution of a random variable [4]. The state of the system at any time is given by a random variable x. The probability distribution $P_x(s)$ gives the likelihood of the system being in state s. With time this probability distribution evolves and is denoted by $P_x(s, t)$. A linear evolution rule for a stochastic system is given in terms of a *Markov chain* for cases where the outcome depends only on the state of the system at the previous time. This transition probability does not depend explicitly on time and can be written as

$$P_x(s(t)|s(t-1))$$

Assuming that there is no loss from the system, the transition probability must satisfy

$$\sum_{s(t)} P_x(s(t)|s(t-1)) = 1$$

For a finite number of states, the probability distribution at any time t can be written as a vector, with each element of the vector corresponding to a state and having value $P_x(s)$. Then the transition probability $P_x(s(t)|s(t-1))$ can be written as a matrix T_x. For a Markov chain, the stationary distribution corresponds to a probability distribution over the state space such that $T_x.P_x^s = P_x^s$.

3.4 Separation of time-scales, metastability and abstraction

The methods used to obtain macroscopic descriptions of systems have their origins in thermodynamics and statistical mechanics. Statistical quantities such as temperature and pressure arise out of the kinetic motion of molecules of the object being observed, and provide a macroscopic description. Similarly, consider a real-world system described macroscopically as a glass of water placed over a table. As time goes by, the water will evaporate from the glass, the glass itself will flow and cover the table and part of it may sublime into vapour. Such high-level descriptions average over spatial locations of individual molecules. In general, a glassy dynamical system can be described in terms of fast, local, quasi-equilibrium dynamics superposed over a slow, nonequilibrium drift [9]. Over a given period of time, the fast motion covers a region of the state space of the dynamical system which can be pictured as a *metastable state*. The slow drift can then be described in terms of

transitions between metastable states. The separation of time-scales into slower and faster processes is based on the time resolution of observation and the total time of observation. The processes that occur in a system can be classified [15] broadly into:

- **Fast processes** that are faster than the time resolution of observation.
- **Slow processes** that are slower than the time resolution of observation.
- **Dynamic processes** that occur on the same time scale of observation.

Macroscopic parameters are averages over the fast processes. These macroscopic parameters that describe the slow processes estabilish a framework for state abstraction.

Any macroscopic description assumes that all parameters of the system described by slow processes are fixed, and that equilibrium applies to all the fast processes.

For an ergodic dynamical system, the states of slower processes correspond to an aggregation of states of faster processes, and the dynamics of slower processes are averages over the dynamics of faster processes [4]. The separation of time scales into fast and slow processes can be used to induce spatial and temporal abstractions over the system. As an example, consider a game of hockey played between two teams. As seen by a player in the game, the dynamics of the game is very complex with the state of the system being described by locations of individual players and the strategy followed by each player. Over a longer time frame, the same game can be described in terms of the score of each team. At this time frame, individual player movements may not be relevant. The dynamics of the game can be described in terms of the change of team scores. This description is much simpler, despite complicated microscopic dynamics. At an even longer time scale, a single game can

be described in terms of the identity of the winning team. This separation of time scales leads to simple macroscopic descriptions.

Each higher level state corresponding to a macroscopic description is an aggregation of many microscopic states. Macroscopic transitions, then, correspond to transitions between aggregations of microscopic states. Since such macroscopic transitions correspond to a slow process, and microscopic transitions to a fast process, macroscopic transitions are relatively rare at a microscopic view of the dynamical system. The dynamics of the system can then be described as largely confined to *metastable* regions of the state space, with relatively few transitions between such regions. An informal definition of metastable behavior constitutes a process such that the time spent in appearing almost in equilibrium in a region of the state space is much larger than the time taken to transition between such regions. Bovier et al. [11] provides a precise definition of metastability for Markov chains by using the separation of time-scales.

The relation of metastability to the spectral characteristics of transition matrices has been studied earlier. Some of the more recent work include Bovier et al. [11], Hartfiel and Meyer [24], Hassin and Haviv [25], Nadler et al. [41, 42], Schütte [48].

Even in cases of non-ergodic dynamical systems, macroscopic parameters are identified as those that are conserved by the fast dynamics of the system.

The separation of time-scales may be violated in some non-ergodic dynamical systems with some dynamical processes occuring at multiple time scales.

3.5 The random-walk as a dynamical system

The evolution rule f^t in a continuous domain is often the solution of a *differential equation of motion*

$$\frac{\partial}{\partial t}x = g(x)$$

Continuous *Brownian motion* can be constructed on any Riemannian manifold and is a diffusion process [22, 54]. This diffusion is described by the heat diffusion equation

$$\frac{\partial}{\partial t}u = -\Delta u$$

The Brownian motion in the continuous case has the following discrete counterpart, the random walk:

$$Pu_n(x) = \sum_y P(x,y)u_n(y) = u_{n+1}(x)$$

where P is the one step transition operator of the walk. A random walk is defined on a graph such that the edge weights define a probability of transition P. In terms of a discrete Laplacian operator,

$$\Delta = I - P$$

and the difference operator in time

$$\partial_n u = u_{n+1} - u_n$$

the discrete heat equation can be written as

$$\partial_n u = -\Delta u$$

The transition rule P forms a discrete evolution map such that

$$u_{n+t}(x) = P^t u_n(x)$$

In general, a random-walk operator T can be used instead of a transition matrix P to obtain a discrete evolution map for a dynamical system. Further, for any random-walk operator T there exists a corresponding random-walk Laplacian $\Delta^{(rw)} = I - T$, and vice versa.

The random-walk on a graph corresponds to a stochastic dynamical system. The random-walk operator T forms a transition matrix for forward evolution. Numerical spectral-analysis of this transition operator can be used to identify metastable regions in the state space, and hence provide a computational basis for state abstraction.

CHAPTER 4

Spatial abstraction

The previous chapter described the connections between abstraction and metastability in dynamical systems. This chapter uses these ideas to provide a novel framework for state abstraction in Reinforcement Learning. Given an RL task formulated as an MDP, a random-walk operator is constructed that is independent of the task nature but reflects the topology of the state space. The random-walk operator is generalized over a group of tasks sharing the same topology. Viewing this random-walk as a stochastic dynamical system provides a basis for task-independent state abstraction. A numerical spectral analysis algorithm, PCCA+, is used to identify metastable regions in the state space, thereby generating a state abstraction.

4.1 A Reinforcement Learning Problem and its related random-walk operator

Consider a discrete Markov Decision Process (MDP) $M = (S,\ A,\ P,\ R)$ with of a finite set of discrete states S, a finite set of actions A, a transition model $P(s, a, s')$ specifying the distribution over future states s' when an action a is performed in state s, and a corresponding reward model $R(s, a, s')$, specifying a scalar cost or reward [44]. The set of actions possible at a state s is given by $A(s)$. The state-space can be modeled as a weighted graph $\overline{G} = (\overline{V},\ \overline{W})$. The states S correspond to the vertices $\overline{V}$ of the graph. The edges of the graph are given by $\overline{E} = \{(i, j) :$

$\overline{W}_{ij} > 0\}$. Two states s and s' are considered adjacent if there exists an action with non-zero probability of transition from s to s'. This adjacency matrix A provides a binary weight ($\overline{W} = A$) that respects the topology of the state space [36]. The weight matrix on the graph and the adjacency matrix A will be used interchangeably from here onwards.

A *random-walk* operator [36] can be defined on the state space of an MDP as $T = D^{-1}A$ where A is the adjacency matrix and D is a *valency matrix*, i.e., a diagonal matrix with entries corresponding to the degree of each vertex (i.e., the row sums of A).

The graph Laplacians are symmetric matrices that are used to study undirected graphs. In cases where the graphs are directed or have asymmetric weights, their properties are approximated in terms of symmetric graph-Laplacians such as $(T + T^\star)/2$ or $T.T^\star$ [13, 19]. $T^\star$ represents the time-reversal of T given by $T^\star(x, y) = \phi(y)T(y, x)/\phi(x)$, where $\phi T = T$. These approximations are used to study properties of directed graphs in terms of closely related undirected graphs. For the purpose of solving the Perron cluster eigenproblem, explained later in Section 4.3.1, the computationally simpler approximation of $(T+T^t)/2$ was empirically found to yield reasonable results.

In case the goal states of the MDP are defined as absorbing states, then these absorbing states are excluded from the graph $\overline{G}$ to get a subgraph G. The absorbing states of the MDP are represented by the set δG and form the *boundary* of the graph G. The sets G and δG are disjoint and $\overline{G} = G \cup \delta G$. The graph G has a weight matrix W which is a restriction of $\overline{W}$ to the vertices G. This weight matrix W corresponds to the Dirichlet boundary condition and defines a random walk *killed* at exiting the set G [54]. The random-walk operator created from this weight matrix

forms a *transition operator* on the graph G and will be denoted by $T|_G$.

4.2 Learning a global random-walk operator over multiple RL tasks

Consider a set of RL tasks that use a common state space $\mathbb{S}$ and share its topology. In general an RL task might be defined only over a subset $S \subseteq \mathbb{S}$. Even for RL tasks defined over $\mathbb{S}$, the goal states might be absorbing and transitions from absorbing states for this task should not be used to estimate the random walk operator for $\mathbb{S}$. The definition of the random-walk operator in the previous section avoids this error by excluding the absorbing states. Therefore the weighted graph $\mathcal{G}$ and the random walk operator $T|_{\mathcal{G}}$ for the entire state space $\mathbb{S}$ can be estimated in terms of the random walk operators $T|_G$ for individual tasks.

A global adjacency matrix $A|_{\mathcal{G}}$ can be maintained corresponding to the global state space $\mathbb{S}$. The adjacency matrix for a given task is a submatrix of $A|_{\mathcal{G}}$. The transitions observed while performing individual tasks are used to update adjacency information into $A|_{\mathcal{G}}$. When each state in $\mathcal{S}$ has appeared in atleast one task, the construction of $A|_{\mathcal{G}}$ is complete. This assumes that in the process of solving a task, every possible transition allowed in that state space is visited atleast once. This is a reasonable assumption to make for most learning algorithms. The complete adjacency matrix can be used to compute the valency matrix D and thereby obtain the random-walk operator $D^{-1}A$. Therefore, the adjacency information alone need be maintained to compute the appropriate random-walk operator.

4.3 The PCCA+ algorithm

The random-walk operator $T|_{\mathcal{G}}$ (shortened to T from here onwards) provides the probability distribution over the state space after a single step of diffusion, and corresponds to a transfer operator over probabilities. T represents a transfer operator or infinitesimal generator of a diffusion on the graph. The existence of a relation between the eigenvalues of such stochastic matrices and metastability has been mentioned earlier.

Recent work in the area of conformational molecular dynamics has lead to robust algorithms to identify metastable states. We use one such algorithm, PCCA+ [17, 56] to automatically identify metastable states. The random-walk operator constructed here can be considered as a discretized spatial transition operator [48] needed for the PCCA+ algorithm. The PCCA+ algorithm identifies metastable regions in the state space by solving a Perron cluster eigenproblem on the transfer operator.

The definitions given below follow the work by Deuflhard and Weber [17], Weber [56].

4.3.1 The Perron cluster Eigenproblem

Given a stochastic matrix T of dimension N, say. Then the eigenproblem for the Perron eigenvalue $\lambda_1 = 1$ has the form

$$\phi^t T = \phi^t, \quad Te = e, \quad \phi^t e = 1$$

In terms of the underlying Markov chain, the left eigenvector $\phi^t = (\phi_1, \ldots, \phi_N)$ can be interpreted as the *discrete invariant measure*, while the right eigenvector

$e^t = (1, \ldots, 1)$ represents the characteristic function of the *discrete invariant set*. Given $u, v \in \mathbb{R}^N$ and the diagonal matrix $\mathcal{D}^2 = \mathrm{diag}(\phi_1, \ldots, \phi_N)$, a special inner-product and its induced norm are defined as

$$\langle u, v \rangle_\phi = \sum_{i=1}^{N} u_i \phi_i v_i = u^t \mathcal{D}^2 v$$

$$\|v\|_\phi = \langle v, v \rangle_\phi^{1/2}$$

to be called the *ϕ-product* and *ϕ-norm* further on.

Uncoupled Markov chains

Let $\rho = \{1, 2, \ldots, N\} = \rho_1 \oplus \cdots \oplus \rho_k$ denote the total index set decomposed into k disjoint index subsets, each of which represents a state subspace. The Markov subchain corresponding to this subspace could be thought of as running over an "infinitely long time", i.e. the subspace corresponds to an *invariant subset*. With the invariant subsets identified the total transition matrix T can be permuted to be strictly *block diagonal* with sub-matrices $T_1, \ldots, T_k$. Each of these sub-matrices is stochastic and gives rise to a single Perron vector $\lambda(T_i) = 1$, $i = 1, \ldots, k$. Assume the sub-matrices to be primitive. Then, due to the Perron-Frobenius theorem, each block T_i possesses a unique right eigenvector having unit entries over the index set ρ_i. In terms of the total $N \times N$ matrix T, the Perron eigenvalue is k-fold, i.e.,

$$\lambda_1 = \cdots = \lambda_k = 1$$

and the corresponding eigenspace is spanned by the extended vectors

$$\chi_i^t = (0, \ldots, 0, \underbrace{1, \ldots, 1}_{\rho_i}, 0, \ldots, 0), \qquad i = 1, \ldots, k$$

These eigenvectors can be interpreted as *characteristic functions* of the discrete invariant subsets. The number of invariant subsets k will be called the *Perron index*. Any Perron eigenvector basis $X = [X_1, \ldots, X_k]$ can be written as a linear combination of the characteristic functions and vice versa. If $\chi = [\chi_1, \ldots, \chi_k]$ a non-singular (k, k)-matrix $\mathcal{A}$ exists such that

$$\chi = X\mathcal{A}, \quad X = \chi\mathcal{A}^{-1}$$

This can be used to arrive at a set of k^2 equations for the k^2 unknowns representing the components of $\mathcal{A}$.

For $\mathcal{A}^{-1} = (a_{ij})$ the components of X can be obtained as

$$X_i(l) = \sum_{j=1}^{k} a_{ji}\chi_j(l), \quad i = 1, \ldots, k, \quad l = 1, \ldots, N$$

which is an over-determined system of kN equations for the k^2 unknowns a_{ji}. Since the Perron eigenvectors are constant on each index subset ρ_m, represent each such subset by a single index $l_m \in \rho_m, m = 1, \ldots, k$ and arrive at the system of k^2 equations

$$X_i(l_m) = \sum_{j=1}^{k} a_{ji}\chi_j(l_m), \quad i = 1, \ldots, k, \quad l = 1, \ldots, k$$

Since $\chi_j(l_m) = \delta_{m,j}$, this system can be directly solved to yield

$$a_{mi} = X_i(l_m), \quad i = 1, \ldots, k, \quad m = 1, \ldots, k$$

or, in matrix notation

$$\mathcal{A}^{-1} = \begin{pmatrix} X_1(l_1) & \cdots & X_k(l_1) \\ \vdots & & \vdots \\ X_1(l_k) & \cdots & X_k(l_k) \end{pmatrix}$$

Nearly uncoupled Markov chains

Consider the subsets ρ_i to form k nearly uncoupled Markov chains such that the probability of transition between any pair is very low. Each such Markov chain could be considered to run "for a long time." Such subsets are therefore called *almost invariant subsets*. In this case the transition matrix $\widetilde{T}$ will be *block diagonally dominant* after a suitable permutation. As a perturbation of the k-fold Perron root $\lambda = 1$, a *Perron cluster* of eigenvalues

$$\widetilde{\lambda}_1 = 1, \quad \widetilde{\lambda}_2 = 1 - \mathbf{O}(\epsilon), \quad \ldots, \quad \widetilde{\lambda}_k = 1 - \mathbf{O}(\epsilon)$$

will arise, where $\epsilon > 0$ denotes a perturbation parameter scaled here as

$$\epsilon = 1 - \widetilde{\lambda}_2$$

The stochastic matrix can be given an ϵ-expansion as follows

$$\widetilde{T}(\epsilon) = T + \epsilon T^{(1)} + \mathbf{O}(\epsilon^2),$$

The Perron cluster eigenvectors $\widetilde{X} = [\widetilde{X}_1, \ldots, \widetilde{X}_k] \in \mathbb{R}^{N \times k}$ can also be ϵ-expanded as

$$\widetilde{X}_i(\epsilon) = X_i + \epsilon {X_i}^{(1)} + \mathbf{O}(\epsilon^2),$$

4.3.2 Almost characteristic functions

The PCCA+ algorithm approaches the inconsistency introduced by $\epsilon \neq 0$ by defining *almost characteristic functions* $\widetilde{\chi}(\epsilon) = [\widetilde{\chi}_1, \ldots, \widetilde{\chi}_k]$ such that

$$\widetilde{\chi}(\epsilon) = \widetilde{X}(\epsilon)\widetilde{\mathcal{A}}(\epsilon)$$

or component-wise, in terms of the k^2 coefficients $\widetilde{\mathcal{A}} = (\alpha_{ij})$,

$$\widetilde{\chi}_i = \sum_{j=1}^{k} \alpha_{ji}\widetilde{X}_j$$

These almost characteristic functions are assumed to satisfy the *positivity* property

$$\widetilde{\chi}_i(l) \geq 0, \quad i = 1, \ldots, k, \quad l = 1, 2, \ldots, N,$$

and the *partition of unity* property

$$\sum_{i=1}^{k} \widetilde{\chi}_i(l) = 1, \quad l = 1, 2, \ldots, N.$$

The *positivity* property and the *partition of unity* property imposed on $\widetilde{\chi}$ defines a $(k-1)$ simplex $\sigma^{k-1} \subset \mathbb{R}^k$. The k vertices of this simplex are the coordinate unit vectors.

4.3.3 The solution

The identification of the Perron cluster (corresponding to $\lambda_i \approx 1$) could be viewed as a linear mapping from eigenvector data into a simplex structure. A *min-Chi* indicator χ_{min} shows deviation from simplex structure and is used to identify the

size of the Perron cluster. The PCCA+ algorithm is then used to compute the almost characteristic functions $\widetilde{\chi}$. This can then be used to induce a partitioning on the state space. A state s is assigned to an *abstract state* or *metastable conformation* numbered $C_s = \text{argmax}_{k=1}^{\chi_{min}} \widetilde{\chi}_{sk}$. A more robust approach would be to run an *inner simplex algorithm* [57] to cluster the states using $\widetilde{\chi}$.

Let χ correspond to a discretized characteristic function obtained using any of these methods. PCCA+ also computes a *coupling matrix* $\widetilde{W}$ [17] reflecting the probabilities w_{ij} of transition between two abstract states i and j. Deuflhard and Weber [17], Weber [56], Weber et al. [57] provide details regarding the actual implementation of the algorithm.

4.3.4 Comparison with other spectral clustering methods

The classical approach to clustering data involves assuming data points to lie in some $\mathbb{R}^n$ space, with distance between points measuring similarity. This Euclidean assumption on the data may not always mirror the intrinsic similarity measure within the data. A more sophisticated approach assumes each data point to be a vertex in a weighted graph $G = (V, W)$ with the edges given by $E = \{(i, j) : W_{ij} > 0\}$. The weights between two vertices correspond to the similarity between the data points corresponding to those vertices. Clustering the data into partitions can be viewed as removing edges from the graph to obtain k disconnected components. The association of a vertex with a component in this disconnected graph induces a k-partition on the vertices. The set of edges is called a *cut* on the graph, and a weight function or *cut-value* can be defined based on the weights on edges belonging

to the cut-set

$$cut(A, B) = \sum_{\{i,j\} \in E, i \in A, j \in B} w(i, j)$$

The min-cut algorithm [59] minimizes the cut-value to partition the graph. The cut-value is independent of the size of the partition, and using it can lead to unbalanced partitions.

An alternate measure, the *average cut*, given by

$$Averagecut(A, B) = \frac{cut(A, B)}{|A|} + \frac{cut(A, B)}{|B|}$$

normalizes on the partition size, and leads to balanced partitions.

The normalized-cut [49] measure normalizes the cut-value on the connectivity within the nodes in a partition. It is given by

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}$$

where $assoc(A, V) = \sum_{\{i,j\} \in E, i \in A, j \in V} w(i, j)$.

The $Ncut$ problem can be solved approximately by identifying real-valued indicator functions for membership to a partition. The eigenvectors of the random-walk Laplacian $\Delta^{(rw)}$ or the normalized Laplacian $\Delta^{(norm)}$ [49, 57] can be used as indicator functions. Some methods [43, 49] obtain k-partitions on the graph by using k-means clustering on these indicator functions and are not robust enough, as shown by Weber et al. [57].

PCCA+ rephrases the partitioning problem in terms of Markov chains to obtain a geometric interpretation of eigenvector data as a simplex. The deviation from simplex structure, measured as a *Min-Chi* value, identifies the number of clusters k

that the data can be partitioned into.

A few methods [29, 49] use the second-largest eigenvalue to successively 2-partition the data to identify k-partitions. Such algorithms may work only if the data has a well-separated simplex structure [55].

4.4 Interpretation

The abstraction obtained on the state space depends on the time-scale at which the agent observes the environment. This section links this time-scale of observation to an informal concept of the size of an agent. An ability to observe the environment at different time-scales can lead to generation of hierarchies of abstraction on the state space. The almost characteristic functions are interpreted as indicating fuzzy membership to abstract states.

4.4.1 The size of an agent

Consider two roof-tops connected by a tightrope. Only a tightrope walker walking with measured, almost deterministic steps would be able to navigate such an obstacle and move between the roof-tops at will. Any other person will find a tightrope as a considerable hindrance or bottleneck during an aimless stroll along the city roof-tops.

Consider a person trying to walk around a ship during a storm. Doors between rooms would hinder navigation due to the rolling and yawing of the ship. Moving around the ship in calmer seas will be much easier.

The examples given above indicate a relation between the granularity or control over movement of an agent and the ability of state-space features to act as bottlenecks. To better understand this relation, a notion of the *size* of the agent

is introduced. Consider bottlenecks as regions in the state space that hinder the movement of the agent. Then the *size* of an agent relative to its environment can be defined informally as inversely related to the probability of transition to a bottleneck.

Simple dynamic systems can be described in terms of processes that occur at different time scales. An agent that interacts with and observes the environment builds a representation of the system based on its time-scale of observation. Consider the state-space of an MDP as a discretization of some real-world domain. This discretization averages over microscopic locations to define discrete states. Smaller agents have a finer discretization on the state space and have a more detailed view of the dynamics of the environment. Larger agents, on the other hand, have a more granular view of the world around them. The size of the agent determines the time scale over which an agent observes the environment. Larger agents have a longer time-scale of observation and identify relatively slow processes that occur in the environment. Identifying the metastable regions at the time-scale of observation induces a spatial abstraction. If the metastable regions in the state space are ordered in terms of the stability of their corresponding attractors then fast processes, corresponding to small agents, are affected by only the most stable attractors. As the time-scale of observation increases, even relatively unstable attractors impact the dynamics of the agent.

The intuition related to the size of an observer indicates that smaller agents will differ from larger agents in their view of the environment. A spatial feature that may appear as a door to a person might appear as a wide corridor to an insect. Therefore the size of an agent decides whether features in the state space act as bottlenecks. Features that may hinder a larger agent may not appear as bottlenecks to smaller agents.

4.4.2 Constructing hierarchies

The size of an agent, decided by its time-scale of observation, defines its view of the environment and identifies the bottlenecks in the environment. A small agent, with fine granularity of movement, views the entire state space as a single abstract state. With increase in the size of the agent, more spatial-features present themselves as bottlenecks to the agent leading to a finer partitioning on the state space. Abstract states identified by an agent are partitioned further by bottlenecks identified by a larger agent. This process induces a hierarchy on abstract states.

This process can be compared to a process of *simulated annealing* on the state space where the temperature is inversely related to the size of the agent. Bottlenecks correspond to peaks on the energy surface and abstract states to regions between these peaks.

4.4.3 Abstract states as Fuzzy sets

Consider an agent standing at the door between two rooms. The agent could be considered to belong to both the rooms. For locations near a door, the agent can claim membership to the adjoining room, but to a lesser extent. The farther the agent gets from the door, the greater the extent of membership to the current room. There is a distance from the door at which the agent truly belongs to the current room. This distance depends on the ability of the door to act as a bottleneck. This in turn depends on the size of the agent. The larger the agent the lesser will be the distance needed from a bottleneck for the current abstract state to completely claim the agent. Consider an obese person having squeezed through a narrow door-way to move to a new room. Such a person is unlikely to return to the previous room even from a location near the door-way.

The almost characteristic functions $\widetilde{\chi}$ resulting from the PCCA+ algorithm can be interpreted as characterizing almost invariant fuzzy sets [17]. For every abstract state S, a state i could be assigned a degree of membership given by $\mathcal{F}(S) : i \rightarrow [0, 1]$. The value of $\widetilde{\chi}$ corresponding to a particular abstract state could be used as $\mathcal{F}(S)$. In cases where clustering techniques are used to allocate states to abstract states a confidence measure could be used to obtain $\mathcal{F}(S)$.

4.5 Experiments

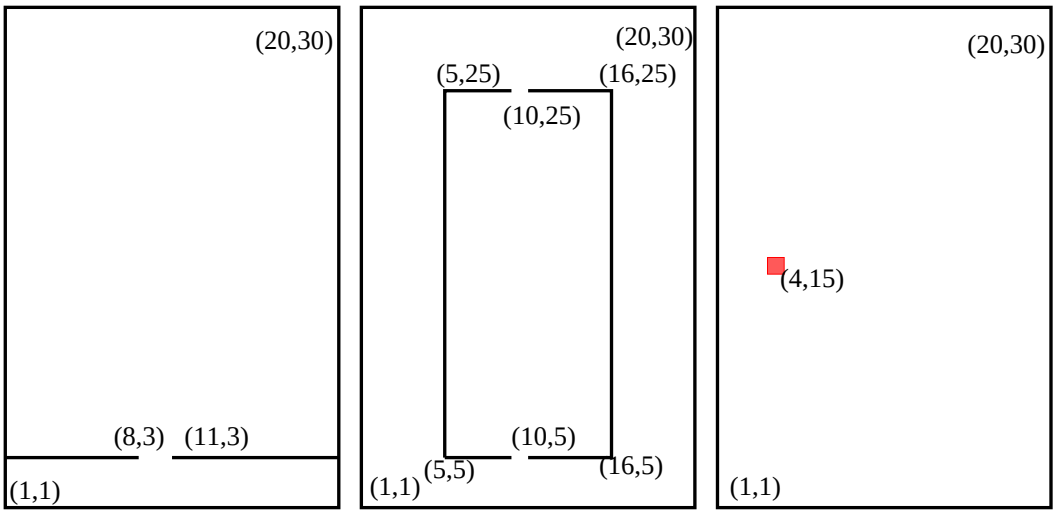
A few simple grid world domains are used to experimentally explore some aspects of the state abstraction algorithm.

4.5.1 Design of experiments

The experiments were designed to validate the state abstraction algorithm against the following

- **Asymmetry**: A few existing state-partitioning algorithms are either biased toward equal-sized partitions [37], or differ from topological notions of bottlenecks [38]. The first experiment was designed to verify the algorithm for asymmetrically sized partitions on the state space.

- **Global structure**: Some automated state-abstraction algorithms [50] restrict themselves to localized searches for *access states*, and do not identify cases where bottlenecks may be related. The global structure of the state space alone can provide this relation. The second experiment consists of two rooms connected by two-doors placed, with the doors placed far apart. A topological



(a) Two rooms **(b)** Room within a room **(c)** Object in a room

Figure 4.1: Test Domains

abstraction would identify two rooms. In this abstraction, it would not matter which door caused a transition between the two rooms.

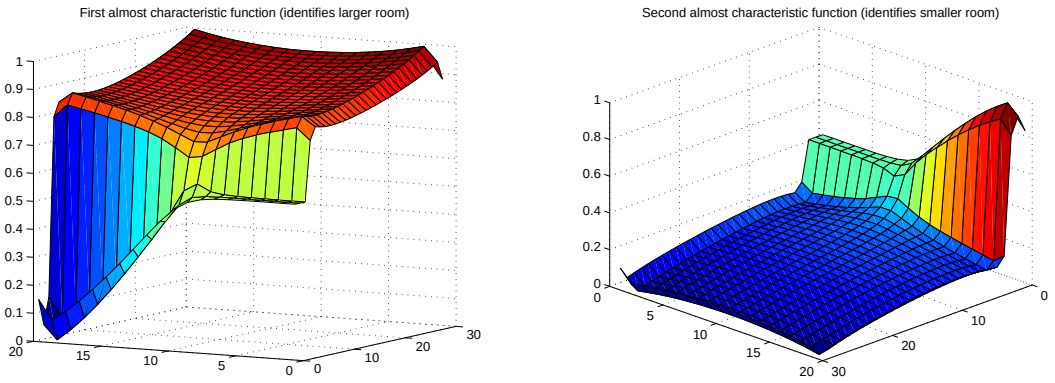
- **Separation of time-scales among state variables**: A few methods find state abstractions by identifying state variables that change at a slower rate than other state variables. The third experiment consists of an object in a room, with a state variable indicating whether or not the agent is holding the object. This experiment validates the proposed abstraction method against such forms of state representation.

Random walk operators over 3 different domains were constructed and abstract states identified. All domains correspond to a 20×30 state grid-world. An agent performing a random-walk in any of these domains has an equal probability of reaching any of its topological neighbors. In most states there are 4 reachable neighbors leading to a probability of 0.25 of reaching a given neighbor.

4.5.2 The two room grid-world

This grid-world domain (Figure 4.1a) consists of two rooms. The first room lies between coordinates (1,1) and (20,3). The second room lies between (1, 4) and (20, 30). A two-state wide doorway between the two rooms allows transitions between them.

Inspite of a large difference in room sizes, the PCCA+ algorithm identified the two rooms. $\widetilde{\chi}_1$ (Figure 4.2a) identifies the larger room and $\widetilde{\chi}_2$ (Figure 4.2b) identifies the smaller room.



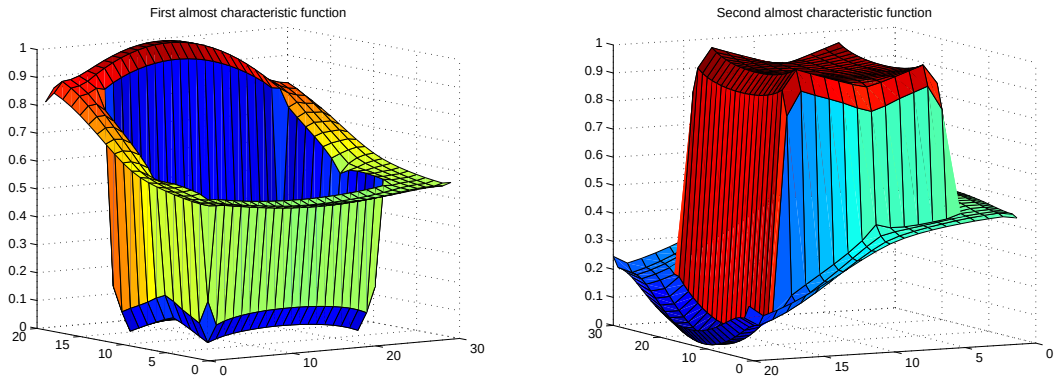
(a) $\widetilde{\chi}_1$ **(b)** $\widetilde{\chi}_2$

Figure 4.2: Two-rooms

4.5.3 A room within a room

This grid-world (Figure 4.1b) consists of an inner room bounded between coordinates (5, 5) and (16, 25). Single-state wide doors to the outer room are provided at locations (10, 5) and (10, 25).

Identifying the two rooms is difficult due to the existence of more than one door, and door placements at topologically distant locations. With the initial dynamics



(a) $\widetilde{\chi}_1$ **(b)** $\widetilde{\chi}_2$

Figure 4.3: Room within a room

of the agent (corresponding to a size of almost 4, due to 4 neighbors for any state), no partitions were identified. To measure the impact of increased size of the agent, the probability of transitions to the doors was decreased. When the probability of the transition was decreased to 0.08, the algorithm successfully identified the two distinct rooms with $\widetilde{\chi}_1$ (Figure 4.3a) identifying the outer room, and $\widetilde{\chi}_2$ (Figure 4.3b) identifying the inner room.

4.5.4 Object in a room

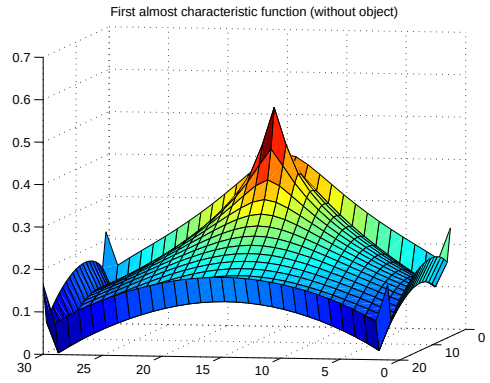
This grid-world (Figure 4.1c) has an object placed at (4, 15). If an agent reaches this state, it automatically picks up the object. The state space is defined in terms of the tuple $(HoldingObject, x, y)$ where the first feature indicates whether the agent is currently holding the object or not.

The state space consists of two abstract states, one with the agent not holding the object and another with the agent holding it. These two abstract states are connected by a directed transition from states near (4, 15) that are not holding the object to the state (4, 15) and holding the object.

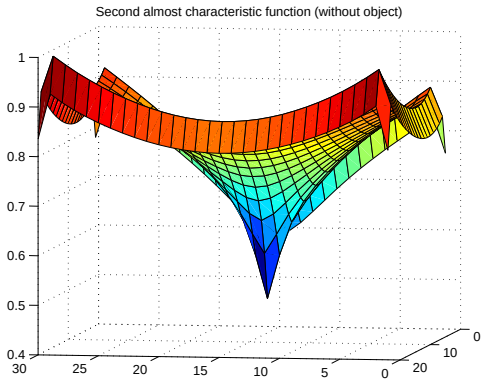
The random walk operator on the directed graph corresponding to this state space is approximated by an undirected graph. This approximation differs from the original dynamics in a probabilistic dropping of the object on transitions out of (4, 15), leading to the neighborhood of $(HoldingObject, 4, 15)$ increasing from 4 to 8 (4 with $HoldingObject$, and 4 with $\overline{HoldingObject}$).

The algorithm identified the two clusters corresponding to the two abstract states $HoldingObject$ and $\overline{HoldingObject}$. For states corresponding to $HoldingObject$, $\widetilde{\chi}_1 > \widetilde{\chi}_2$ (Figures 4.4c and 4.4d). Similarly, for the abstract state $\overline{HoldingObject}$, $\widetilde{\chi}_2 > \widetilde{\chi}_1$ (Figures 4.4b and 4.4a). Therefore the almost characteristic function $\widetilde{\chi}_1$ corresponds to $HoldingObject$ and $\widetilde{\chi}_2$ to $\overline{HoldingObject}$. Three states with transitions to the location (4, 15) were incorrectly classified as belonging to the abstract state $HoldingObject$. The simplex structure correctly classified all states.

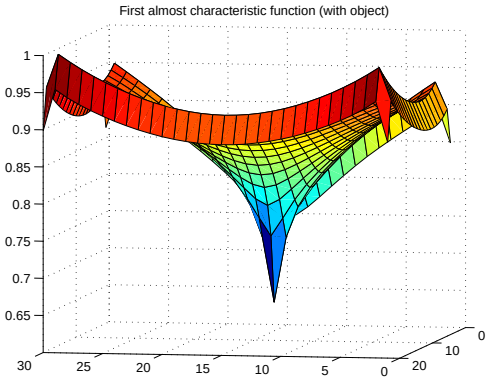
The state $(HoldingObject, 4, 15)$ has eight neighbors in the undirected graph approximation. Other states have a maximum of only 4 neighbors. The comparatively larger neighborhood helps in identifying the state transition as a salient event, and thus identify the partition.



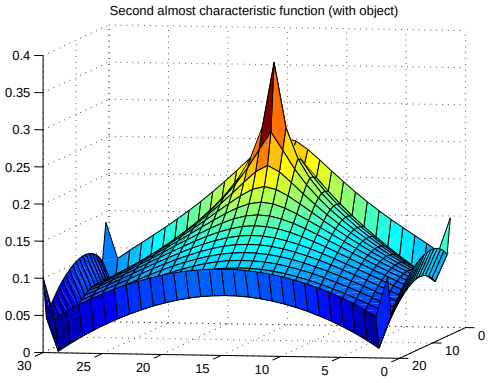
(a) $\widetilde{\chi}_1$(Without object)



(b) $\widetilde{\chi}_2$(Without object)



(c) $\widetilde{\chi}_1$(With object)



(d) $\widetilde{\chi}_2$(With object)

Figure 4.4: Object in a room

CHAPTER 5

Temporal abstraction

The dynamics of an agent can be described as largely confined to metastable abstract states with comparatively fewer transitions between such abstract states. The idea of combining task-independent state abstraction with temporal abstraction is connected with the Intrinsically Motivated RL framework. A few techniques to improve the performance of abstraction and skill-reuse are discussed. In particular, combining abstraction with the proto-value function framework has computational advantages.

Temporally-extended actions can be constructed to model transitions between abstract states. These extended actions are useful because they define low probability transitions and can be used to improve sampling of the state space. The Options framework is used to model transitions between abstract states as *subtask options*.

5.1 Automatic construction of Subtask Options

Consider an option $O = \langle \mathcal{I}, \pi, \beta \rangle$ in an MDP $\mathcal{M} = \langle S, A, P, R \rangle$ as defined earlier. For a subtask corresponding to a traversal between two abstract states given by the state spaces $S_1, S_2 \subseteq \mathbb{S}$, an *augmented* option is constructed. Consider an *augmented* option $O_{S_1 \to S_2} = \langle \mathcal{I}, \pi, \beta, R \rangle$ defined over a *reward-free MDP* $M = \langle S, A, P \rangle$ where $S \supseteq S_1 \cup S_2$.

The rewards related to a *subtask option* can now be associated with the subtask.

This *intrinsic* reward can be designed to learn the subtask either by a reward on termination and a discounted return formulation for cumulative reward, or by any other suitable means. A *global* reward-free MDP $\mathbb{M} = \langle \mathbb{S}, \mathbb{A}, \mathbb{P} \rangle$ can be defined where $\mathbb{S}, \mathbb{A}$, and $\mathbb{P}$ are part of a global environment.

The coupling matrix $\widetilde{W}$ (Refer Section 4.3.3) identifies abstract states that are connected to each other. The transitions between two connected abstract states can be stored as a subtask option. For every transition $S_1 \rightarrow S_2$, a subtask option $O_{S_1 \rightarrow S_2} = \langle \mathcal{I}, \pi, \beta, R \rangle$ can be defined with $\mathcal{I} = S_1$ and $\beta = \chi(S_2)$.

The undirected graph approximation for directed graphs can lead to the creation of spurious subtask-options. As an example, the two automatic options constructed in the object-in-a-room experiment (Section 4.5.4) correspond to picking up the object *PickUpObj* and dropping the object *DropObj*. The *DropObj* option results due to the undirected graph assumption. A generic agent will not be able to learn a policy for such subtasks and therefore they will never be added to the knowledge-base of a generic learning agent.

5.2 Intrinsic Motivation

Curiosity drives human-beings to understand the environment around them. In the process of interacting with the environment and exploring it, often accomplishing mundane tasks such as driving to work or walking around in a building, we construct abstract models of the environment. These models are built independent of specific tasks, but mirror the environment. Such an internal representation of the world remains consistent across tasks and enables us to reuse skills across different tasks. The skills that we learn over time enable us to gain greater control over the

environment in the long term.

The framework of Intrinsically Motivated Reinforcement Learning (IMRL) [5, 51] looks at constructing a set of reusable skills using notions of saliency or novelty of stimuli. The options-framework [53] is used to learn skills corresponding to *salient events* in the environment. The implementation of such an Intrinsically-Motivated agent [51] assumes a hard-coded notion of salient events. Further, beyond adding all options to the set of admissible agent actions, there is no attempt to filter out relevant skills which may be applicable to a particular task. For an agent with a large knowledge-base of reusable skills stored as options, adding all options as admissible actions when only a few may be actually needed, will end up slowing the learning process of the agent.

For an implementation to be practical, not only should task-independent skills be identified automatically, but there also exists a need to filter out irrelevant skills while solving a particular task. The abstraction methods presented here extend the IMRL framework to achieve these goals.

5.2.1 Task-independent abstraction

An RL task is specified over an environment by adjusting rewards and defining goal states in an MDP. The global random-walk operator defined earlier is constructed independent of both rewards and goal states. Therefore, the abstract states identified by running PCCA+ on this operator are also independent of the task. As an example, the two-room grid world (Section 4.5.2) is partitioned into two rooms independent of the location of the goal state or the reward distribution. All tasks that differ in their reward distribution alone but have the same goal state would share the same random walk operator and would thus lead to the same state abstraction.

Tasks that differ in the location of their goal state would still share the same global random walk operator. The construction of the complete global random walk operator using random walk operators of individual tasks leads to state abstraction over the global state space shared by all these tasks.

5.2.2 Salient events and temporally extended actions

The state abstraction technique presented identifies metastable regions in the state space. Transition between two such metastable regions is a relatively rare event. Such transitions are abstracted as subtask options. A subtask takes an agent to a significantly different region of the state space, and the transition involves novelty to the agent. This task independent notion of novelty which depends on the state space alone fits into the Intrinsically Motivated RL framework. Further such subtasks are identified automatically without preconceived notions of salient events. The object-in-a-room experiment (Section 4.5.4) indicates that useful skills such as picking up objects can also be identified by the algorithm. This experiment indicates that the state abstractions are not just geometrical in nature, but also capture some notion of saliency.

5.3 Relevant subtasks and approximately policies

The dynamics of the agent can be viewed in terms of abstract states and subtask options that correspond to transitions between connected abstract states. To solve any task that shares the same dynamics, the spatial and temporal abstractions can be used to obtain a high-level policy over abstract states.

Consider the case of tasks with only a single goal state, and no other absorbing

states. A single successful trajectory obtained can be used to identify the goal state and its corresponding abstract state. The *coupling matrix* $\widetilde{W}$ provides connectivity information over these abstract states. Due to the symmetric nature of the random-walk operator and the construction of the state abstraction, the coupling-matrix is also symmetric. Using the connectivity information alone, $\widetilde{W}$ provides an adjacency matrix over the abstract states considered as an undirected graph. Given the abstract state corresponding to the goal state, this information can be used to prune irrelevant subtasks from the knowledge-base.

If the abstract states are considered to form an undirected graph, then a policy over these states corresponds to associating directions to the edges. If the search is restricted to deterministic optimal policies, then we can apply the following constraints:

- No edges start from the goal state.
- Only a single edge leads out of every non-goal state. This edge corresponds to the optimal action from that state.
- The goal state is reachable from all states.

The optimal policy therefore corresponds to a tree rooted at the goal state. All such trees can be enumerated from the undirected graph to generate a set of deterministic policies.

The rewards specific to the task can be sampled to associate cumulative-rewards with each subtask option. Value iteration or Dijkstra-like algorithms can then be used to find optimal policies over abstract states.

As an example, consider the two-room domain (Section 4.5.2). The abstract state representation of the environment corresponds to two states *UpperRoom* and

LowerRoom connected to each other. The subtask-options identified would be *MoveToUpperRoom* and *MoveToLowerRoom*. This representation could be viewed as a two-vertex undirected graph. Assume that the subtask options have already been trained and are available in a knowledge-base of useful skills. Given any task corresponding to a single goal state, an optimal policy over the abstract states involves associating a direction to the single edge in the graph. If the goal is in the upper room, then all states in the lower room could be initialized to use the *MoveToUpperRoom* subtask option. In a more complex domain such as an apartment complex, the solution will not be as straightforward. An abstract view of the environment can be represented as a graph with vertices corresponding to rooms, and edges between vertices corresponding to interconnected rooms. Given a goal state corresponding to a room, an optimal policy involves identifying each room with a single optimal inter-room transition. Further, these transitions should be selected so that a path exists from any room to the destination room. No transitions lead out of the destination room. Therefore, the process of assigning a direction to each vertex converts the graph into a tree rooted at the destination vertex. Identifying such trees restrict the space of possible solutions and makes it easier to solve the problem. To identify the policy over abstract states, the graph edges could be associated a weight corresponding to the path cost. The value function associated with the goal-state of every subtask-option provides an indication of the path cost associated with each temporal abstraction. This weighted graph could then be analyzed using Dijkstra's algorithm or value-iteration to determine the policy over abstract states.

The subtask options have an intrinsic reward structure and may not share the same reward structure as the task specified. Therefore the policies associated with subtasks may not be optimal under the reward structure of the task, but may have

approximate optimality due to the similarity in the nature of the subtask. Given a task and its related reward structure, subtask options can be modified to use the reward structure of the given task to produce new task-specific options. These new options reflect the reward structure provided by the task and therefore policies constructed over them will be recursively optimal.

5.4 Spatio-Temporal abstraction and the Proto-value function framework

The proto-value function framework [34, 36] uses the eigenvectors of the graph Laplacian as an orthonormal basis for approximating value-functions. The Representation Policy Iteration (RPI) algorithm [35] combines the automatic construction of an orthonormal basis with the Least Squares Policy Iteration (LSPI) [31] algorithm. These eigenvectors are the same as that of the random-walk operator. Calculating the first k-eigenvectors of the graph Laplacian is the most computationally intensive part of the abstraction algorithm, a task shared with the proto-value function framework. Therefore if the implementation of PCCA+ is merged with an agent running the RPI algorithm, then the state space abstraction can be obtained with a relatively low increase in computational cost. A brief outline of such an algorithm is provided below in Algorithm 5.4.1.

A computationally efficient algorithm to identify spatial abstractions when combined with the acquisition and reuse of useful skills can lead to a generic learning agent with broad competence.

for each task do

Representation Learning Phase:

Learn an adjacency matrix $A|_G$ over the state space of the task and obtain a random-walk operator T. $A|_G$ may also be extracted from the global adjacency matrix $A|_{\mathcal{G}}$.

Update the global $A|_{\mathcal{G}}$ using the task-specific $A|_G$.

Compute the top k eigenvectors of T and collect them as a basis-function matrix ϕ.

Identify number of abstract states using the *minChi*-indicator.

Run the PCCA+ algorithm to partition the state-space.

Identify skills using the state-partition and the coupling-matrix $\widetilde{W}$, and add them to skill-KB.

Control Learning Phase:

Sample trajectories can be used to identify the goal-state.

Identify relevant skills using this goal-state information and $\widetilde{W}$ to obtain approximate recursively-optimal policies, thereby bootstrapping the learning process.

for each transition $(s_t, a_t, s'_t, a'_t, r_t)$ **do**

for all subtask options $O_{S_i \to S_j}$ in skill-KB with s_t, s'_t in the state-space of the option **do**

Use transition information and *intrinsic* reward to update Q-value for option.

Update basis-weights w^i for ϕ as indicated in the RPI algorithm.

Obtain an approximate optimal value function $Q = \sum_i w^i \phi$, as described in the RPI algorithm, for the current task.

Algorithm 5.4.1: Pseudo-code for a generic learning agent

CHAPTER 6

Conclusion and Future work

This thesis provides a mathematical framework to state-abstraction in Reinforcement Learning by using ideas from dynamical systems. The separation of time-scales in simple dynamical systems leads to a high-level view of the dynamics in terms of metastable regions and transitions between them. A random-walk extracted from an MDP is modeled as a dynamical system and used to generate a state abstraction based on metastability.

The task-independent nature of the random-walk operator causes this abstraction to be independent of both rewards and the location of the goal state specified in the MDP. Transitions between metastable regions are identified as skills and added to a knowledge-base for reuse in other tasks. An ability to look beyond the nature of the immediate task at hand ensures that the skills identified are task-independent. Such skills depend on the topology of the state-space alone and can generalize across tasks that share a common random-walk operator. Further, the nature of skills identified in this thesis correspond to relatively rare transitions between metastable regions. Such skills can help an agent explore the state space more effectively.

The task-independent nature of state abstraction and the novelty associated with the temporal abstraction are used to connect the proposed method with the Intrinsically Motivated RL framework. The proposed method improves over the original framework by its ability to automatically determine salient or novel events.

For simple problems, where the goal state is the only absorbing state, a pruning method can filter out irrelevant subtasks before identifying an optimal policy over

abstract states. The subtasks can further be tuned to the reward structure of any particular task. These ideas are similar to work on Discounted-Reward TSPs [10, 33], but extended to stochastic domains.

Combining the proposed spatial-abstraction method with the Representation Policy Iteration algorithm from the Proto-value function framework adds only a small computational overhead. Such an algorithm can be used as a basis to construct generic agents.

Future directions

The global random-walk operator constructed here uses adjacency information alone and disregards actual probabilities of transition. Domains such as the windy grid worlds correspond to drifting random walks over a uniformly random policy. This bias will not get reflected in the adjacency information. In such cases, the transition probabilities provide a more accurate reflection of the environment. The amount of training data required to accurately determine transition probability is much more than for adjacency information. A study of the tradeoffs between suitable approximations and sample complexity might be useful.

The random-walk operator is expected to be symmetric, with approximations required for other cases. Symmetric adjacency operators lead to approximations when

- Two neighboring states have different number of neighbors. As an example, in a grid world domain, a state near a wall may have only three neighboring states, while other states may have four neighbors.
- The dynamics provided by adjacency information may be irreversible. As an

example, a transition might exist from one state to another, but the reverse transition which could take the agent back to the original state might not exist.

The approximation suggested earlier for asymmetric adjacency matrices give good results for simple examples. There is a need for improved approximations, especially in the case of irreversible dynamics. Alternate methods for state abstraction that do not need symmetric operators might lead to more generic algorithms.

The size of the agent is defined informally, and is determined by the transition probabilities on the state space. A formalized definition of agent-size, along with a way to simulate various sizes will provide a way to construct state space hierarchies.

The abstraction methods presented here are task independent. Useful locations within abstract states could be identified by observing reward structures from the given task. Temporal abstractions restricted to regions within an abstract state could provide a finer granularity on the state space.

The global random-walk operator constructed here depends on the topology of the state space alone. The Bellman operator could be used to obtain a localized distance metric over the state space. Since such a transfer operator would include reward information, the generated spatial abstraction would be reward based.

Consider environments where the optimal-policy for a subtask option is guaranteed to succeed. In other words, a trained subtask-option $O_{S_1 \to S_2}$ will always take the agent from an abstract state S_1 to S_2. At a high level, the dynamics of the agent can then be described in terms of abstract states and *deterministic* transitions between them. Due to the nature of policies at this level, methods from deterministic

planning domains can be used to identify policies. A simple pruning method to identify relevant subtasks was provided for domains with a single absorbing state. There might exist more sophisticated methods that can be applied to complex domains.

All environments might not be amenable to deterministic descriptions. A simple example would be a grid-world with doors to different rooms positioned near each other and sufficient stochasticity so as to be unable to provide guarantees on transitions between rooms. A study of deterministic approximations to stochastic domains might be useful.

In general, an intrinsically motivated agent is likely to incur a cost for building broad competence while solving a given task. With the cost of computational power spiraling downward, there might be a trend toward building advanced knowledge-bases able to learn concepts automatically and reuse them across different tasks. Notions of similarity between concepts or tasks might become more sophisticated and lead to more generic learning agents.

REFERENCES

[1] Agre, P. E. (1988). The Dynamic Structure of Everyday Life. Technical Report AITR-1085, Massachusetts Institute of Technology, Cambridge, MA, USA.

[2] Agre, P. E. and Chapman, D. (1987). Pengi: An implementation of a theory of activity. In *Proceedings of AAAI-87*, pages 268–272, Menlo Park, California. AAAI Press.

[3] Amarel, S. (1968). On representations of problems of reasoning about actions. In Michie, D., editor, *Machine Intelligence 3*, volume 3, pages 131–171. Elsevier, Amsterdam.

[4] Bar-Yam, Y. (1997). *Dynamics of Complex Systems*. Addison-Wesley.

[5] Barto, A. G., Singh, S., and Chentanez, N. (2004). Intrinsically Motivated Learning of Hierarchical Collections of Skills. In Triesch, J. and Jebara, T., editors, *Proceedings of the 2004 International Conference on Development and Learning*, pages 112–119. UCSD Institute for Neural Computation, The Salk Institute for Biological Studies, La Jolla, California, USA.

[6] Belkin, M. and Niyogi, P. (2003). Laplacian Eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396.

[7] Belkin, M. and Niyogi, P. (2004). Semi-supervised learning on Riemannian Manifolds. *Machine Learning*, 56(1-3):209–239.

[8] Berger, M. (2002). *A Panoramic View of Riemannian Geometry*. Springer.

[9] Biroli, G. and Kurchan, J. (2000). Metastable states in glassy systems. arXiv:cond-math/0005499v2.

[10] Blum, A., Chawla, S., Karger, D. R., Lane, T., Meyerson, A., and Minkoff, M. (2003). Approximation algorithms for Orienteering and Discounted-Reward TSP. In *FOCS '03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*, page 46, Washington, DC, USA. IEEE Computer Society.

[11] Bovier, A., Eckhoff, M., Gayrard, V., and Klein, M. (2000). Metastability and small eigenvalues in Markov chains. *Journal of Physics A: Mathematical and General*, 33(46):L447–L451.

[12] Bradtke, S. J. and Duff, M. O. (1995). Reinforcement learning methods for continuous-time Markov decision problems. In Minton, S., editor, *Advances in Neural Information Processing Systems*, volume 7, pages 393–400, Cambridge, MA. MIT Press.

[13] Chung, F. R. K. (2005). Laplacians and the Cheeger inequality for directed graphs. *Annals of Combinatorics*, 9:1–19.

[14] Coifman, R. R. and Lafon, S. (2006). Diffusion Maps. In *Applied and Computational Harmonic Analysis: special issue on Diffusion Maps and Wavelets*, volume 21, pages 5–30. Springer-Verlag.

[15] Cvitanović, P., Artuso, R., Mainieri, R., Tanner, G., and Vattay, G. (2005). *Chaos: Classical and Quantum*. Niels Bohr Institute, Copenhagen. `ChaosBook.org`.

[16] Dean, T. and Lin, S.-H. (1995). Decomposition techniques for planning in stochastic domains. In *Proceedings of the 1995 International Joint Conference on Artificial Intelligence*.

[17] Deuflhard, P. and Weber, M. (2005). Robust Perron Cluster Analysis in Conformation Dynamics. In Dellnitz, M., Kirkland, S., Neumann, M., and Schütte, C., editors, *Special Issue on Matrices and Mathematical Biology*, volume 398C of *Linear Algebra and its Applications*, pages 161–184. Elsevier.

[18] Ding, C. and He, X. (2004). Linearized cluster assignment via spectral ordering. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 30, New York, NY, USA. ACM Press.

[19] Fill, J. (1991). Eigenvalue bounds on convergence to stationarity for nonreversible Markov chains, with an application to the Exclusion process. *The Annals of Applied Probability*, 1(1):62–87.

[20] Finney, S., Wakker, P., Kaelbling, L., and Oates, T. (2002). The thing that we tried didn't work very well : Deictic representation in reinforcement learning. In *Proceedings of the 18th Annual Conference on Uncertainty in Artificial Intelligence (UAI-02)*, pages 154–161, San Francisco, CA. Morgan Kaufmann Publishers.

[21] Givan, R., Dean, T., and Greig, M. (2003). Equivalence notions and model minimization in Markov decision processes. *Artificial Intelligence*, 147(1-2):163–223.

[22] Grigor'yan, A. (1990). Analytic and geometric background of recurrence and non-explosion of the Brownian motion on Riemannian Manifolds. *Bulletin of American Mathematical Society*, 36:135–249.

[23] Grigor'yan, A. (2006). Heat kernels on Weighted Manifolds and applications. In *The Ubiquitous Heat Kernel*, volume 398 of *Contemporary Mathematics*, pages 91–190. AMS.

[24] Hartfiel, D. J. and Meyer, C. D. (1998). On the structure of stochastic matrices with a subdominant eigenvalue near 1. *Linear Algebra and its Applications*, 272(1–3):193–203.

[25] Hassin, R. and Haviv, M. (1992). Mean passage times and nearly uncoupled Markov chains. *SIAM J. Discret. Math.*, 5(3):386–397.

[26] Hein, M., Audibert, J.-Y., and von Luxburg, U. (2006). Graph Laplacians and their convergence on random neighborhood graphs. Available at arXiv:math.ST/0608522.

[27] Jonsson, A. and Barto, A. (2006). Causal Graph Based Decomposition of Factored MDPs. *Journal of Machine Learning Research*, 7:2259–2301.

[28] Jonsson, A. and Barto, A. G. (2000). Automated state abstraction for Options using the U-Tree algorithm. In *Advances in Neural Information Processing Systems 13*, pages 1054–1060.

[29] Kannan, R., Vempala, S., and Veta, A. (2000). On clusterings-good, bad and spectral. In *FOCS '00: Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 367, Washington, DC, USA. IEEE Computer Society.

[30] Knoblock, C. A. (1990). Learning abstraction hierarchies for problem solving. In Dietterich, T. and Swartout, W., editors, *Proceedings of the Eighth National Conference on Artificial Intelligence*, Menlo Park, California. AAAI Press.

[31] Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *J. Mach. Learn. Res.*, 4:1107–1149.

[32] Land, M., Mennie, N., and Rusted, J. (1999). The roles of vision and eye movements in the control of activities of daily living. *Perception*, 28(11):1311–1328.

[33] Lane, T. and Kaelbling, L. (2002). Nearly deterministic abstractions of Markov Decision Processes.

[34] Mahadevan, S. (2005a). Proto-value functions: developmental reinforcement learning. In *ICML '05: Proceedings of the 22nd international conference on Machine learning*, pages 553–560, New York, NY, USA. ACM Press.

[35] Mahadevan, S. (2005b). Representation policy iteration. In *Proceedings of the 21th Annual Conference on Uncertainty in Artificial Intelligence (UAI-05)*, pages 372–37, Arlington, Virginia. AUAI Press.

[36] Mahadevan, S. and Maggioni, M. (2006). Proto-value Functions: A Laplacian Framework for learning Representation and Control in Markov Decision Processes. Technical Report TR-2006-45, University of Massachusetts, Department of Computer Science.

[37] Mannor, S., Menache, I., Hoze, A., and Klein, U. (2004). Dynamic Abstraction in Reinforcement Learning via clustering. In *ICML '04: Proceedings of the twenty-first International Conference on Machine learning*, volume 21, pages 560–567, New York, NY, USA. ACM Press.

[38] McCallum, A. K. (1995). *Reinforcement Learning with Selective Perception and Hidden State*. PhD thesis, University of Rochester, New York.

[39] McGovern, A. (2002). *Autonomous Discovery of Temporal Abstractions from Interaction with an Environment*. PhD thesis, University of Massachusetts.

[40] Minut, S. and Mahadevan, S. (2001). A reinforcement learning model of selective visual attention. In *AGENTS '01: Proceedings of the fifth international conference on Autonomous agents*, pages 457–464, New York, NY, USA. ACM Press.

[41] Nadler, B., Lafon, S., Coifman, R., and Kevrekidis, I. (2006a). Diffusion Maps, Spectral Clustering and Eigenfunctions of Fokker-Planck Operators. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems 18*, pages 955–962. MIT Press, Cambridge, MA.

[42] Nadler, B., Lafon, S., Coifman, R., and Kevrekidis, I. (2006b). Diffusion Maps, Spectral Clustering and reaction coordinates of Dynamical Systems. In *Diffusion Maps and Wavelets*, volume 21 of *Applied and Computational Harmonic Analysis*, pages 113–127. Elsevier.

[43] Ng, A. Y., Jordan, M. I., and Weiss, Y. (2001). On spectral clustering: Analysis and an algorithm. In Dietterich, T. G., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems 14*, pages 849–856. MIT Press.

[44] Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley-Interscience.

[45] Ravindran, B. and Barto, A. G. (2002). Model Minimization in Hierarchical Reinforcement Learning. In *Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation*, pages 196–211, London, UK. Springer-Verlag.

[46] Ravindran, B. and Barto, A. G. (2003a). Relativized Options: Choosing the Right Transformation. In *Proceedings of the Twentieth Internatinal Conference on Machine Learning (ICML 2003)*, pages 608–615, Menlo Park, CA. AAAI Press.

[47] Ravindran, B. and Barto, A. G. (2003b). SMDP Homomorphisms: An Algebraic Approach to Abstraction in semi-Markov decision processes. In *Proceedings of the Eighteenth Internatinal Joint Conference on Artificial Intelligence (IJCAI 2003)*, Menlo Park, CA. AAAI Press.

[48] Schütte, C. (1999). *Conformational Dynamics : Modelling, Theory, Algorithm, and Application to Biomolecules*. Habilitation thesis, Dept. of Mathematics and Computer Science, Freie Universität Berlin.

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

[50] Şimşek, Ö., Wolfe, A. P., and Barto, A. G. (2005). Identifying useful subgoals in reinforcement learning by local graph partitioning. In *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005)*, pages 816–823. ACM.

[51] Singh, S., Barto, A. G., and Chentanez, N. (2005). Intrinsically Motivated Reinforcement Learning. In Saul, L. K., Weiss, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems 17*, pages 1281–1288. MIT Press, Cambridge, MA.

[52] Sutton, R. and Barto, A. (1998). *Reinforcement Learning: An Introduction*. MIT Press.

[53] 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.

[54] Telcs, A. (2006). *The Art of Random Walks*, volume 1885 of *Lecture Notes in Mathematics*. Springer.

[55] Weber, M. (2004). Clustering by using a simplex structure. Technical Report ZR-04-03, Zuse Institute Berlin (ZIB).

[56] Weber, M. (2006). *Meshless Methods in Conformation Dynamics*. PhD thesis, Department of Mathematics and Computer Science, Freie Universität Berlin.

[57] Weber, M., Rungsarityotin, W., and Schliep, A. (2004). Perron Cluster Analysis and Its Connection to Graph Partitioning for Noisy Data. Technical Report ZR-04-39, Zuse Institute Berlin.

[58] Whitehead, S. D. and Ballard, D. H. (1991). Learning to Perceive and Act by Trial and Error. *Machine Learning*, 7:45–83.

[59] Wu, Z. and Leahy, R. (1993). An optimal graph theoretic approach to data clustering: theory andits application to image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(11):1101–1113.

[60] Zhu, X., Ghahramani, Z., and Lafferty, J. (2003). Semi-supervised learning using Gaussian fields and Harmonic Functions. In *ICML '03: Proceedings of the twentieth International Conference on Machine learning*.

[61] Zorich, V. A. (2004a). *Mathematical Analysis I*. Springer-Verlag.

[62] Zorich, V. A. (2004b). *Mathematical Analysis II*. Springer-Verlag.

LIST OF PAPERS BASED ON THESIS

1. B. Ravindran, Andrew Barto, and Vimal Mathew (2007). Deictic Option Schemas. *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 1023–1028.