GENERALIZED REINFORCEMENT LEARNING WITH AN APPLICATION IN GENERIC WORKLOAD MANAGEMENT SYSTEMS

by

PO-HSIANG CHIU

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON
August 2011

ACKNOWLEDGEMENTS

The research experience at UTA and Brookhaven National Laboratory has one way or another provided many life situations, human connections that support my mental, emotional and spiritual growth. I truly believe in the intelligent design of the universe which, albeit not in a linear and straightforward sense, unfolds life events in a perfect sequence that leads me to work with the best people on or off campus and that also helps me in finding the nicest friends who have given me lots of emotional support.

First, I am very grateful to my mentor Dr. Manfred Huber, who has been supportive in my perhaps slightly unconventional way of research that at times may appear to be outside of the safety zone in bringing forth the information. Manfred has been a strong teacher and a great source of ideas that have contributed greatly in establishing the work presented in this dissertation. I would also like to thank my former thesis advisor Dr. Diane Cook. I appreciate her guidance and feedback in the early stage of this research. Also thanks to David Levine, Gergely Zaruba, Jaehoon Yu, Ramez Elmasri for their inputs as my committee members.

I am also truly grateful for the collaborative research opportunity at Brookhaven National Laboratory presented to me by Mr. Levine and Dr. Yu. The interdisciplinary research environment in the lab certainly stimulated creative thinking necessary to establish the work in this dissertation. I have also partly fulfilled my passion in physics through the discussions and meetings with great staff members and researchers in the lab. For this, I would also like to thank Dr. Torre Wenaus for his valuable feedbacks in the research and development of Grid applications. Special thanks also to Jaime from the Condor team at University of Wisconsin at Madison for his assistance in technical issues related to the Condor system. Thanks also to Jose, Prem, Sudhamsh, Maxim for their assistance in the research work. In the mean time, I would also like to extend my gratitude to the PAS and RACF Groups in the Physics Department

at Brookhaven National Laboratory for their feedback, guidance and assistance in the required software systems and computing facilities.

Last but not the least, I truly appreciate the mental and emotional support from my family and friends. I would not have embarked on this journey and been able to push through it with relative ease without their love and encouragement. Although this chapter of the life journey has come to an end, the experience certainly has been fun, fruitful, and no doubt has served a good foundation for the next adventure.

July 29, 2011

ABSTRACT

GENERALIZED REINFORCEMENT LEARNING

WITH AN APPLICATION IN GENERIC

WORKLOAD MANAGEMENT

SYSTEMS

Po-Hsiang Chiu, PhD

The University of Texas at Arlington, 2011

Supervising Professor: Manfred Huber

Learning by trial and error and being able to form levels of abstraction from the past experience has been an important factor for sentient beings to develop intelligent behaviors and cope with an ever-changing environment. Complex control domains, in a similar way, often require the interacting agents to learn adaptive control strategies for time-varying or potentially evolving systems. This dissertation will begin by investigating an example of complex domains, Grid computing networks, through a collaborative effort in the design and implementation of a generic workload management system, PanDA-PF WMS used in the ATLAS experiment. With the incentive of boosting the performance of PanDA-PF WMS and increasing its applicability in a general resource-sharing environment, we will subsequently motivate an automated and adaptive learning approach that optimizes computational resource usage.

From the experience of developing Grid applications such as PanDA-PF, we found that a flexible infrastructure still has its limit in performance both from the perspective of high-performance computing (HPC) and high-throughput computing (HTC). The key is that an

optimal resource allocation strategy is highly contingent upon many factors hidden in the intricate dynamics behind the scene, including the task distribution and real-time resource profile in addition to compatibility between the user tasks and the allocated machines, etc. The reinforcement learning framework establishes a unique way of solving a wide range of control and planning tasks through the state space representation of the system over which the control policy unfolds as a sequence of control decisions toward a maximum payoff. Intuitively, reinforcement learning seems to be an ideal candidate among machine learning methods for developing an optimal resource allocation strategy that harvests free computation resources by learning their intricate dynamics.

However, our hope in applying standard reinforcement learning in the context of resource allocation is diminished due to an inherent limitation in its representation. In particular, the control policy is often formulated from the perspective of decision theoretic planning (DTP) such that actions, as control decisions, are assumed to be atomic with fixed action semantics. Consequently, the derived policy in general lacks the ability in adapting to possible variations in the action outcomes or the action set itself due to versatility of the system. This would be a major barrier in learning an ideal resource allocation strategy where each compute resource is often characterized by time-varying properties that determine its performance. In addition, the available resource may be highly volatile depending on the resource-sharing infrastructure. In a dynamic computational cluster, for instance, the underlying resource is acquired on-demand in terms of distributed virtual machines that may not be persistently available to end users. As a consequence, the optimal strategy for task assignment learned earlier may not be strictly applicable in the future.

Inspired by the challenge in complex domains like optimal resource sharing, this dissertation will progressively develop an extended reinforcement learning framework with a concept-driven learning architecture that enables adaptive policy learning over the abstraction of the progressively evolved samples of experience. In particular, we provide an alternative view

of reinforcement learning by establishing the notion of the reinforcement field through a collection of policy-embedded particles gathered during the policy learning process. The reinforcement field serves as a policy generalization mechanism over correlated decisions through the use of kernel functions as a state correlation hypothesis in combination with Gaussian process regression as a value function approximator. Subsequently, through "kernelizing" the spectral clustering mechanism, the policy-learning experience retained in the memory of the agent can be further subdivided into a set of concept-driven abstract actions, each of which implicitly encodes a set of context-dependent local policies. We will show from a simulated task-assignment domain that the end result of our generalized reinforcement learning framework will enable both the learning of an action-oriented conceptual model and simultaneously deriving an optimal policy out of the high-level conceptual units. Moreover, to demonstrate the general applicability of our learning approach, we apply the work in a generalized navigation domain – the gridworld without the grid in which the agent is free to move in all directions with stochastic behaviors in actions and subsequently show the learning result in terms of both an improved learning curve and reinforcement field plots.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Machine Learning to Face Grid Computing Challenges

In the scientific community, many research and engineering projects over the past few years have gradually evolved to large-scale collaborations from different organizations through the use of geographically dispersed compute resources over the network. Projects that require such collaborative endeavors often involve cross-disciplinary settings with massive amounts of data exchanged for the purpose of simulation and analysis such as Monte Carlo computing, large-scale optimization, and pattern discovery. Examples can be observed in the ATLAS experiment [82-84], the Human Genome Project [67], and SERENDIP [80], etc. To harness the distributed computational power, often the problem to solve is decomposed into several subtasks, whether it be independent user jobs or workflows containing several interdependent tasks (e.g. DAGMan [59]), followed by allocations to the desired compute resources for their results and feedback. Correspondingly, a user-friendly interface for task submissions, output retrievals, and fast turn-around time are among the core issues to be considered.

Further, it is foreseeable that the conjoint effort of shared and distributed resources will potentially extend beyond the boundaries of current dedicated clusters and computing farms, as seen in the conventional Grid, to the inclusion of sensor networks, personal computers, mobile computers, etc, as evidenced by the growth of Volunteer Computing [53] and Opportunistic Computing [48]. Heterogeneity of distributed compute resources will thus become increasingly prominent. Nevertheless, even with the current Grid environment of modest scale with mostly clouds of computing farms, there already exist highly diversified infrastructures in the aspects of resource brokerage, data transfer, site services, and resource access and sharing mechanisms,

among many others. Thus, this research will first address the optimality of distributed resource integration from the perspective of the design of a generic workload management system.

On the other hand, the efficiency of workload management highly depends on proper resource allocation such that user tasks are always assigned to compatible compute resources in real time. Yet due to the inherent complexity in the task assignment domain especially under a highly volatile resource-sharing environment, the architectural design approach may still be inadequate. As we shall see in the subsequent chapters, a generic workload management system aims to form an abstraction over the complex resource-sharing landscape such that the user only perceives the heterogeneous distributed resources in terms of a uniform set of virtual machines. In this manner, users can effectively utilize the integrated processing power from the resource pool without needing to be concerned about the complexity of the underlying infrastructure. However, due to the nature of the resource-sharing modality, the acquired virtual machines may not be long-lived but will have to be released back to the "ocean" of the distributed resource network (e.g. the Grid or hosts from the Volunteer Computing domain) after a certain period of usage in order to achieve fair-share among users across administrative domains. That is to say, the set of available computational resource does not remain unchanged over time. Even if, under a simpler scenario, the computational resource stays available to end users (e.g. Grid computing farm), the resource profile (e.g. load average) is time-varying as a result of the sharing from multiple users. Consequently, from the perspective of the task scheduler, it is challenging to identify an optimal scheduling policy since the definition of an "optimal policy" needs to be updated constantly in response to the task distribution and the system profile.

The Condor system provides a solution to task scheduling in terms of a match-making process through Condor *ClassAds* [59, 61]. In particular, both user tasks and target machines are associated with user-defined requirements (following the syntax of the ClassAds) such that the requirement has to be mutually agreeable from both ends in order for a match to occur. In

this manner, users can specify their preferences over the computational resource while resource providers also enforce their machine usage policy of interest. However, the user-defined match-making policy like ClassAds may not be objectively defined with the consideration of the actual dynamics of the resource pool due to unavailability of such information (which requires a constant monitoring and analysis) and mutually incompatible interests between users and machine owners in regard to resource sharing. Thus, this research will motivate an adaptive scheduling policy based on automated learning that scavenges appropriate computational resource with sufficient processing power in a manner that goes beyond the limit of man-made control of the policy in order to maximize a particular performance metric of resource sharing (e.g. maximum resource utilization, minimum turnaround time, etc). In particular, predefined match-making criteria such as those defined through the Condor ClassAds can only go as far as finding qualified, compatible resources but will not foresee an optimal match-making policy.

In general, optimal control in complex domains including the task assignment problem described above requires learning an adaptive policy in response to the continuously varying environment such as volatile Grid resources. The reinforcement learning (RL) framework [8] establishes a unique way of deriving such control policies over a state space representation in which the policy unfolds as a sequence of actions (e.g. all the possible task assignments), guiding the agent towards a trajectory with a maximized accumulated reward (e.g. minimum turnaround time for user tasks). However, varying or evolving system dynamics such as the Grid poses a great challenge to mainstream reinforcement learning algorithms due to an inherent limitation in its representation as we shall see shortly. This research further extends the existing RL framework with novel constructs such as the Reinforcement Field (Chapter 6) and the Concept-Driven Learning Architecture (CDLA) (Chapter 7), leading to an extended framework that enables adaptive policy learning and forming abstractions from past experience. This newly-developed architecture is effectively a more generalized RL method that enables the

intelligent agent to develop a transient cognitive model that generalizes the learned policy to the varying and possibly evolving environment. Further, the learned abstraction in CDLA can be shown to be able to represent the match-making criteria necessary for a near-optimal task assignment policy. Moreover, the generalized RL framework in this dissertation has been developed with the consideration of general applicability in other domains as well. For reasons that will become clear later in the subsequent chapters, this research addresses the core issue of the RL formalism rather than merely providing an ad-hoc solution catered for the Grid computing domain. Section 1.2 and 1.3 further details the approach towards implementing an efficient workload management system from two different perspectives; namely, pure architectural design and automatic learning.

1.2 Architectural Design Perspective

In light of the inherently complex and diversified resource-sharing environment, a more generic framework can serve as a foundation to accommodate potentially multiple forms of computing resources. The first half of the research examines the current Grid computing technology, including Globus [64], Condor, and Dynamic Virtual Clusters [65-66] among others. Next, by combining the strengths of multiple systems and though a conjoint effort with the researchers in Physics Department at Brookhaven National Laboratory (BNL), we develop a new workload management system based upon the existing PanDA system with a new extension using the pilot mechanism [71, 86] – Pilot Factory (see Chapter 2). The PanDA architecture is built mostly on top of the existing networking infrastructure and database technology with a user-friendly and uniform interface to task submission and scheduling. The PanDA system works naturally with a general pilot mechanism, a method of resource allocation where pilots are distributed to candidate resources to capture their real-time information, prepare and validate the computing environment before requesting real payloads of pending user jobs. In this manner, users need not be concerned about the details and differences in the

underlying Grid infrastructure such as its associated scheduler and brokerage system; meanwhile, the real-time resource status collected by pilots allows for more robust job scheduling and processing without unexpected failures resulting from, for example, inaccurate estimates of resource capacity (e.g. load average, CPU time, remaining memory), an incomplete software stack, or missing input datasets.

1.3 Learning and Adapting Perspective

A flexible workload management system enables a better integration of the computational resources distributed across administrative domains. However, in order to fully reach the potential of the combined processing power, an adaptive learning framework is often desirable that further breaks the limit imposed by the architecture and human factors associated with the resource sharing mentioned earlier. In this dissertation, we will develop an adaptive task scheduling policy from the perspective of the reinforcement learning framework. However, as we shall see later in Chapter 4, the standard RL formalism would face a great challenge in learning an effective control policy in complex domains such as task scheduling over the Grid resource with non-stationary machine profile and varying availability.

The development of RL has made considerable advancement in recent years largely based on the Markov Decision Process (MDP) [92] and its extension, Partially Observable MDP (POMDP) [93]. RL establishes a unique way of solving a wide range of control and planning tasks through the state space representation of the system over which the control policy is derived. Most formulations in standard RL, however, rely on a prior knowledge of action choices, which are interpreted from the decision-theoretic planning (DTP) perspective as a set of control decisions such as the executable motions using available actuators in robotic control domains. As a consequence, the policy learning for a given task is largely expressed through the features encoding the state space in which the actions collectively serve as a state transition mechanism towards the objective of maximizing the future accumulated reward. The decision

process thus depends on a set of actions, discrete or continuous, that act on a given state in a predefined manner either in a deterministic or stochastic sense. A few immediate implications arise under such a formalism. First, the computational cost of maintaining a global system state can be exceptionally high if not intractable for complex domains. An extreme case can be observed from using a multiagent approach to address decentralized MDP problems [30] such as applying the cooperative multiagent MDP framework in task assignment domains [31]. In such framework, since user tasks and compute resources are represented by their individual state and action spaces, learning a task assignment policy requires forming the join state space as the Cartesian product of the states of all machines and all tasks. The computational cost of maintaining such global system state can be exceptionally high if not intractable for a large Grid network. Second, the control policy is expressed over a concrete set of actions, each of which effectively models an allowable behavior given a situation collectively characterized by related states; however, most policy learning methods do not fully utilize such aggregate states, as addressed by the factored-MPD framework [96,97], where only a consistent subset of actions takes effective steps towards the goal. Consequently, much computational effort is spent in the state space enumeration during the policy evaluation.

On the other hand, performing actions often involves certain behavioral details characterized by a dynamic process contingent upon not only the internal state of the agent but also on the environment. Expressing such a dynamic process, referred to as *action dynamics* in this dissertation, would require a set of parameters being controlled simultaneously. In an automatic vehicle steering task [99], for instance, more than a few physical attributes are involved in specifying a particular choice of motion control (as an action) such as steering the vehicle within a certain range of velocity following the direction expressed in a vector form. Similarly, the task assignment to a particular machine, when modeled as a control decision, highly depends on the real-time resource profile which involves several machine attributes such as the CPU speed and remaining memory, etc. Furthermore, the standard RL framework often

lacks the ability to efficiently adapt to an evolving environment or unexpected behaviors from the action in that the agent is not aware of the "meaning" of actions beyond the policy being followed. These two seemingly different aspects, i.e. action dynamics and policy adaptability, are indeed closely related. Consider a robotic navigation domain, for instance, the varying environment dynamics could be related to a new bumpy surface en route or simply a poor calibration in the motor that occurs only after an optimal policy is learned. As a result, the agent may be deviated away from the course where the optimal policy is feasible without adjusting the policy in parallel. Similarly, variations in the available action set can also alter the agent's ability to perceive and interact with the environment. Using standard RL, performing real-time error recovery for skewed action outcomes and adjusting the learned policy accordingly is not possible without reevaluating the policy from the start since the varying environment effectively redefines the underlying RL problem.

Furthermore, certain features, albeit important to the policy learning, may not be directly accessible to the agent. Examples can be found in distributed systems such as the aforementioned task assignment problem in a workload management system in which the features characterizing the computational resource may not be readily attainable by the agent working remotely to optimize the scheduling policy. We shall see more details in this regard in Chapter 2.

1.4 Dissertation Overview

This dissertation is subdivided into two logical parts. The initial discussion in the next chapter provides, from the Grid architecture perspective, methods for establishing a user-friendly and efficient resource sharing network and finally concludes with a generic workload management system based on the framework of PanDA [85], Condor [55] and distributed virtual machines [60, 68, 71]. The subsequent chapters of the dissertation will shift gear towards a novel RL-based approach for further optimizing the workload management system by providing

a solution based on learning the dynamics of the resource-sharing environment, from which to derive optimal task assignment policy. Subsequently, we will also demonstrate that the new RL framework is also applicable to other domains such as robot navigation.

CHAPTER 2

A GENERIC WORKLOAD MANAGEMENT SYSTEM: PANDA-PF WMS

This chapter will develop a generic workload management system from the perspective of a cleaner architectural design towards more efficient dispatch of workload, resource discovery and processing. In the current Grid environment, efficient workload management presents a significant challenge, for which there are a wide range of de facto standards encompassing resource discovery, brokerage, and data transfer, among others. An immediate question then arises: what would be a flexible framework that glues these diversified Grid components together yet without compromising their interoperability? In addition to a clean architecture, an efficient resource allocation strategy (or equivalently, task assignment strategy) is also essential for boosting the overall productivity of the Grid network or other types of resource-sharing environment in general. However, an ideal resource allocation strategy is subject to variations in the real-time resource profile, which is often not readily accessible without a properly configured Grid infrastructure along with its real-time resource monitoring. In particular, the real-time resource status includes not only static attributes such as CPU speed, total memory, and other system-wise configurations but also time-varying properties such as current available CPU time, load average, and remaining disk space, among others. With the consideration of these interrelated aspects in the resource-sharing landscape, the first goal of this dissertation is to motivate a flexible, generic workload management system (WMS) that also sets the stage for developing an efficient resource allocation strategy. The design of the WMS takes into account several realistic issues that occur in the task assignment. For instance, it is imperative to ensure that the basic computing environment of a machine is well validated before injecting real job payloads in order to minimize unnecessary waste of computing resource.

Empirical observations often indicate that a perfectly functioning compute resource can still fail to accomplish the task as a result of missing pieces of software, staging errors of input files and/or temporary unavailability of network connection, etc. However, the individual machine profile, being remote to the local administrative domain, is usually not directly attainable; that is, the external resource often appears as a black box to the local user. Without reliable real-time status, it is difficult to apply scheduling policies accurately tailored for the job requirements idiosyncratic to different research and engineering projects. As briefly outlined in Chapter 1, the proposed workload management system (WMS) hinges upon the Production And Distributed Analysis System (PanDA) [85, 87] extended by a new pilot dissemination mechanism, i.e. Pilot Factory. For ease of exposition, the new WMS is referred to as PanDA-PF WMS in the later discussion. The PanDA system works naturally with a general pilot mechanism [86], a method of resource allocation where pilots are distributed to candidate resources to capture their real-time information, prepare and validate the computing environment before requesting real payloads of pending user jobs. In this manner, users are shielded from the complexity of the Grid infrastructure. In particular, the dynamically distributed pilots collectively form a layer of abstraction on top of the Grid network so as to present end users a clean, unified view of the underlying computational resources.

The rest of the chapter is organized as follows: Section 2.1 provides an overview of the proposed workload management system based on the PanDA framework. Section 2.2 establishes the related terminology catered to a general resource sharing environment. Section 2.3 briefly reviews the design and current implementation of the PanDA system and its connection with the general pilot mechanism. Section 2.4 covers an overview of Condor and discusses the role of Condor-based pilot factory in resource allocation. Numerical results that compare regular pilot dispatch using Condor-G and the pilot-factory approach using Condor glidein are presented in Section 2.5. Section 2.6 describes the experience with the PanDA system on active scientific experiments. Section 2.7 outlines related work and open issues in

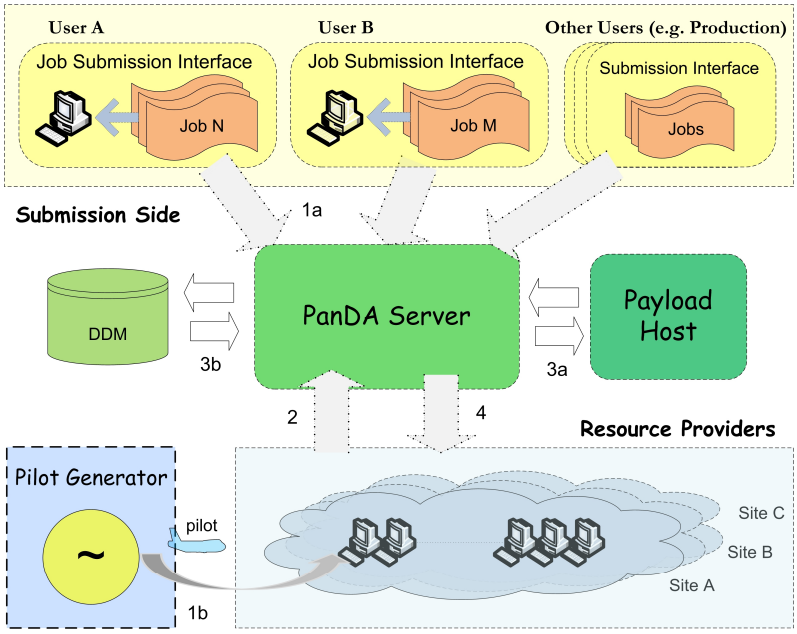


Figure 2.1 A High-level view of the PanDA architecture.

workload management. Lastly, section 2.8 outlines implications of the PanDA-PF WMS and the related research to be continued in the later chapters of this dissertation.

2.1 High-Level View of the Proposed Architecture

The workload management architecture under the PanDA framework achieves resource allocation by systematically sending a series of precursor jobs, namely pilots, to prepare candidate resources before fetching real payloads via PanDA server. Figure 2.1 illustrates the PanDA system when applied to a typical Grid environment with multiple participating sites contributing resources. As shown in the figure, the user submits jobs to PanDA server via a simple client interface where each job defines the associated input and output files, desired matching criteria, and secure channels (e.g. HTTPS, GSIFTP, etc) from which job payloads can be obtained, etc. Note that the payload, in general, refers to any executables required for completing a task. These user jobs are then transmitted to the PanDA server via secure HTTP, authenticated using Grid proxy certificates, followed by returns of

submission status information to the client software. The pilot generator comes into the picture for disseminating pilots periodically to candidate compute resources with an adjustable rate. Optionally, PanDA server communicates with one or more distributed data management systems such as ATLAS DDM [83] to pre-stage input data required for given user jobs. Nonetheless, DDM and details of its data movement are complex subjects in their own rights and are not the primary focus of this dissertation.

As suggested in Figure 2.1, the PanDA architecture follows the separation-of-concerns principle [88] by decoupling job submission, job retrieval, data management, and resource allocation to distributed components. In this manner, the architecture has the advantage of higher adaptability; that is, users are free to choose locally-customized systems or preferable platforms for each role in the flow of workload processing, thereby achieving software interoperability, as is highly preferable in general and complex Grid settings. For instance, pilots can be distributed via Condor-G [60], a Condor system extension that enables jobs to be submitted over the Grid through Globus-enabled gatekeepers [63] that bridge between sites across administrative domains. Other contemporary batch systems such as Glite [73] and PBS [74] are possible alternatives for pilot submissions.

Under the PanDA architecture, efficient job dispatch and resource utilization hinges upon timely resource discovery, job sandboxing, and impromptu status reporting of target machines. To this end, a more scalable and automated version of pilot generator, pilot factory, is developed based on Condor's glidein mechanism [60, 61]. Thus, our goal in the chapter is to finally reach a generic workload management system, namely PanDA-PF WMS, by taking advantage of the properties of PanDA, distributed pilots, and Condor glidein.

2.2 Terminology

Before proceeding to the construction of the PanDA-PF WMS, it is helpful to elaborate some of the terminology used throughput this chapter.

The current resource-sharing environment mainly includes two different categories: i) the Grid Computing environment where the compute resource generally refers to the dedicated computing farms affiliated with certain organizations – such as companies, research labs, etc – that are mutually accountable, and ii) the Volunteer Computing environment in which public processing and storage resources (typically PCs) are combined in an efficient way both architecturally and algorithmically as a conjoint effort to support computational needs from complex projects. In a futuristic sense, the second category can be extended to include any online computational devices such as mobile computers, sensor networks, etc. Since the ultimate goal of the PanDA-PF WMS is to adapt to a general resource-sharing environment regardless the underlying middleware, the party that provides compute resources is generally referred to as a *resource provider*. This is analogous to the site affiliated with particular Virtual Organizations (or VOs) [64] in common Grid literature.

In a computational cluster, the particular server that manages the backend compute resources is referred to as a *head node* (or a front-end node) to be generic; it is analogous to, in the Globus architecture, the gatekeeper machine which typically has a batch system installed such as Condor or PBS in order to further schedule the jobs to the desired backend resources. Such backend resources are sometimes referred to as worker nodes (WNs) in the Grid Computing context [85, 86].

For convenience, *resource boundary* is defined to refer to the set of available compute resources that are reachable from the local administrative domain. Resource boundary morphs as the external resources join or leave the local domain (e.g. Condor glideins to be discussed in Section 2.4). In this dissertation, we refer to the general Grid-type network characterized by a dynamically morphing resource boundary as *metagrid*. A metagrid can be induced by Condor glideins, Dynamic Virtual Clusters (DVCs) [65, 66, 68], PanDA-PF WMS or other mechanism for distributing virtual machines.

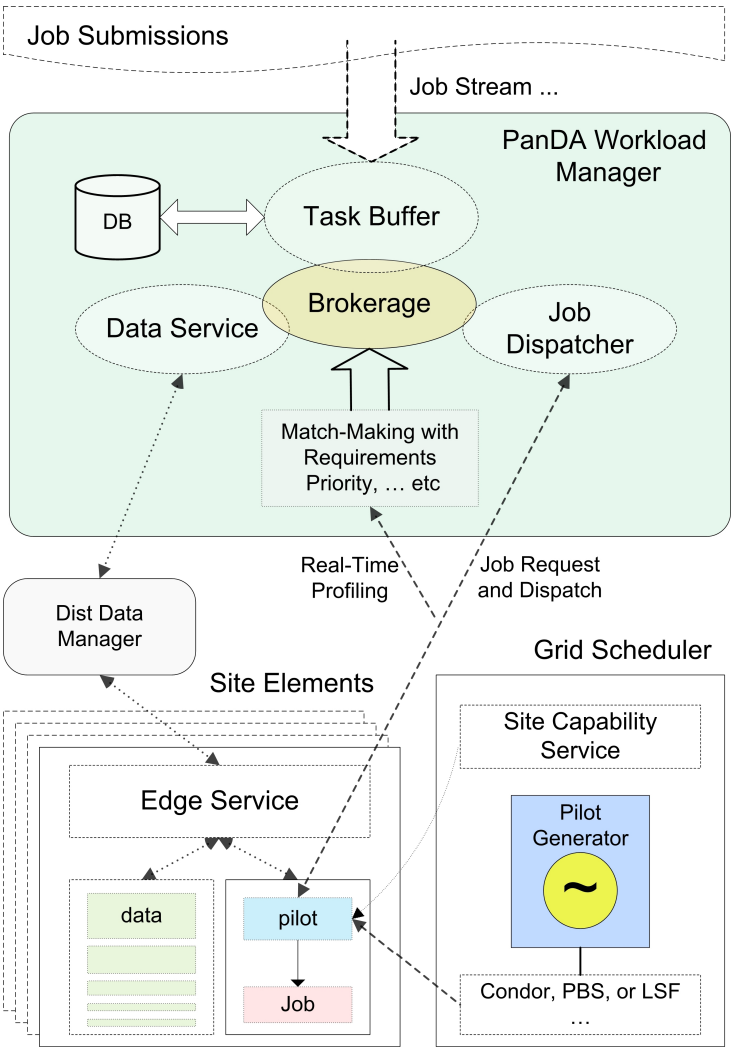


Figure 2.2 Jobs and pilots in PanDA system. Job stream flowing into the PanDA server as it interacts with the external data storage system to pre-stage the required dataset while pilots fetch appropriate jobs to their host resources.

2.3 PanDA and Pilot Generator

Production ANd Distributed Analysis system (PanDA) has been developed since Fall 2005 at BNL to support petabyte-scaled and data-driven production and distributed analysis processing in the ATLAS experiment. The main focus here is to introduce the important features of PanDA directly linked to the architectural benefits as a general WMS, and flexible, on-

demand access to distributed resources through the pilot mechanism. To start with, there are four essential components in the PanDA system as illustrated in Figure 2.2:

- *Task Buffer* represents a job repository containing user job information including related input and output files, various system requirements, job type, priority scheme, location of job's payload, etc.
- *Data Service* interfaces with a distributed data management system (e.g. ATLAS DDM) that performs stage-in and stage-out of the data on which the user job depends.
- *Job Brokerage* is a match-making component that prioritizes and assigns tasks on the basis of known static attributes such as job type, user-defined priority, locality of input data, required resource capacity (e.g. CPU speed, memory, disk space, etc), and other VO- or site-specific brokerage criteria. Job Brokerage in combination with pilots, distributed over candidate resources, completes the desired match-making cycle where pilots further provide dynamic, real-time resource attributes to help with the decision process of job assignments. Different scheduling policies can be supported by Job Brokerage, consistent with the local site's administrative requirements.
- *Job Dispatcher* follows the secure link specified in user jobs and sends job's payloads to the designated compute resources upon pilot requests.

In a generic setting, PanDA Task Buffer functions as a system-wide, attribute-rich job database that records both static and dynamic information on all jobs submitted over the Grid. There is no inherent restriction on the task representation and client interface to PanDA's Task Buffer. In the current implementation, PanDA uses a LAMP stack in which job submission is accomplished via a simple Python- and http-based client without dependency on the underlying Grid middleware. Job specifications are parsed and stored in PanDA's backend database.

Other possible job abstraction schemes for the PanDA front-end client include XML-based job specification, the UDDI framework [79], Condor ClassAds [61], etc.

While jobs are being uploaded to Task Buffer, the pilots are running on the candidate resources, in the process of which pilots make requests to Job Dispatcher in order to obtain job payloads that match with pilot-resident hosts. Behind the scene, the best-fit job is determined by querying Job Brokerage that executes a given match-making algorithm. The decision process is based in part on the user job requirements and preferences and in part on the real-time snapshots that pilots had taken from their hosts. As we shall see in subsequent chapters, this is the key component to be optimized using the automated learning approach. Allocation of jobs is followed by the dispatch of corresponding input data, handled by Data Service, to those pilot-resident hosts; in the meantime, Data Service interfaces with DDM, responsible for data movements, to obtain the desired data. In the PanDA framework, data pre-placement in target machines is ensured before the start of job execution to avoid failures from data staging, which in turn usually results in esoteric failure modes and waste of available computing resource such as CPU time. Data pre-staging is part of the schemes that implement the *late-binding policy* used in the PanDA framework as further discussed in the following subsection.

2.3.1 Pilot

Resource sharing mechanisms in the current landscape of the Grid environment can be classified into two types: one is the *push system* where user tasks are allocated, or pushed, to the available resources via the task scheduler following a given match-making policy; conversely, the other type belongs to the *pull system* where resources, often volunteer hosts like personal computers, initiate connections to the designated task servers, pulling jobs to the hosts where scheduling policies are being executed from within the volunteer's domain. Common batch systems used nowadays (e.g. Condor, PBS [74], LSF [75], etc) in the Grid community mostly falls into the push-system category. Volunteer computing systems such as

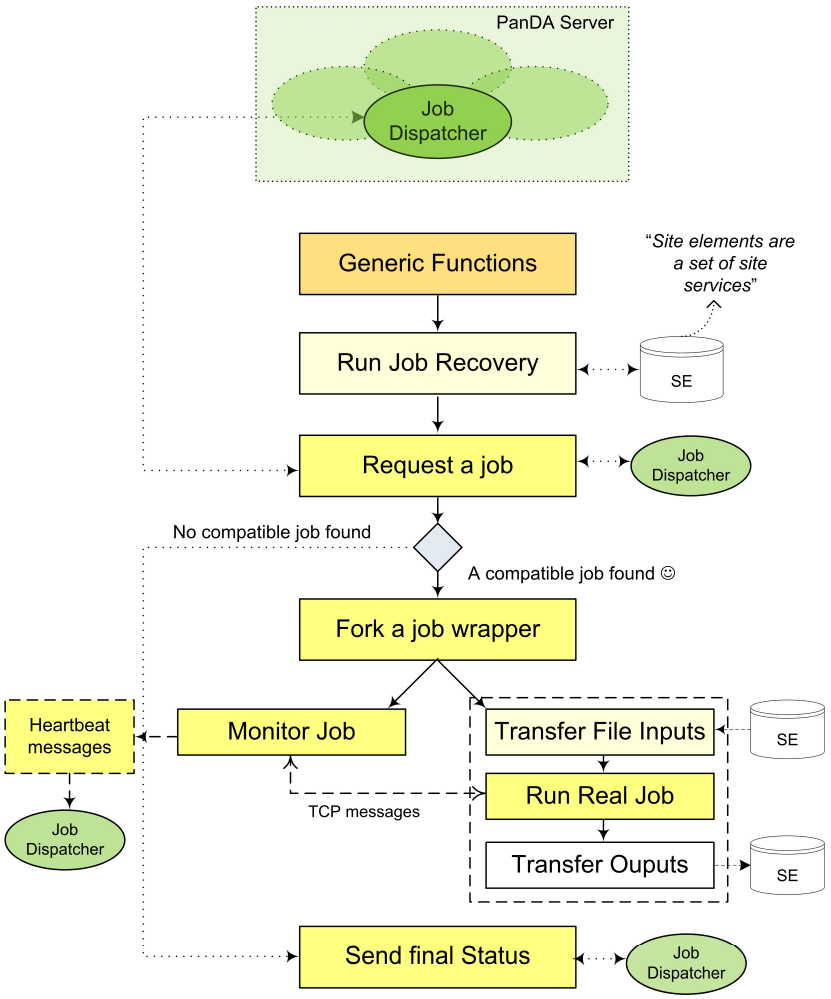


Figure 2.3 Detail schematic of a pilot process. Pilot processes are analogous to light-weight virtual machines. When a pilot becomes active, it pulls a compatible user task from the queue (e.g. from Task Buffer in PanDA via Job Dispatcher) and subsequently runs and monitors the user task. If no compatible tasks exist, then it simply exits, releasing the used resource.

BOINC [51] are mostly considered as pull systems in which compute resources are often occluded behind firewalls or NATs and hence, resource sharing requires initiation from the machine's end.

The PanDA system, on the other hand, is a hybrid system following a so called late-binding strategy in that user jobs are eventually bound to best-fit resources by first pushing

pilots to the candidate resources to perform computing environment provisioning, followed by pulling actual job payloads through the PanDA server.

In the most generic use case, a pilot functions as a light-weight user job that validates the most rudimentary resource properties such as shell environment, interpreter/compiler availability, basic software stacks, system configurations, and network connectivity and additionally, performs real-time resource profiling such as the remaining memory capacity, etc. These checks are performed to secure a basic working environment for the end-user and also provide a snapshot of resource availability used to facilitate an optimal match-making process.

As illustrated in Figure 2.3, pilots can be designed hierarchically with a generic layer that performs only a high-level system-wise validation prior to its immediate binding with the site-specific service layers, which are implemented as a separate pilot core, containing the site's local services such as methods for file transfer, security and sandboxing mechanisms, etc. Once the computing environment checks are completed, pilots then proceed to the following job-specific routines [86]:

1) *Data Transfer*: After receiving the job payload from PanDA server, the pilot invokes Data Service to draw in the required input data from DDM using the site's preferable copy tools (e.g. GSIFTP); pilot also transfers output and log files back to end users upon completion of the designated task. Depending on the implementation scheme, file stage-out can also be performed instead by DDM in response to pilot requests, conforming more strictly to the separation-of-concerns principle.

2) *Job Execution*: Pilot spawns a process as a job wrapper that copies input files, sets up runtime environment, executes the job payload, transfers job output files (either by itself or by delegating to DDM) and then finally performs final clean-ups. Conversely, if no job is received, the pilot simply cleans up the work directory and exits.

3) *Monitoring*: The pilot runs job monitor as a separate thread that tracks the runtime states and packs the information in terms of periodical heartbeat messages back to the Job

Dispatcher at the PanDA server. If Job Dispatcher does not receive the message after a pre-defined period, it will consider that the job had failed and thus notify the pilot to kill the job. Moreover, each job's runtime information is updated accordingly upon receiving heartbeat messages.

4) *Job Recovery*: Temporary unavailability of the resource can often lead to job's valid outputs being stranded at a remote storage system or worse yet, being deleted by clean-up operations from the resource provider itself. Site maintenance, system overhead, or job preemption enforced by a site's local policy could all results in such temporary disconnect. The pilot in this scenario will attempt to rerun the entire file transfer mechanism mentioned earlier; if failed, the same file-transfer operation will then be executed by the successive pilots until a pre-defined limit is reached.

The PanDA system accomplishes the match-making process not only through the static job requirements stored in Task Buffer but also through the dynamic resource attributes published from the pilots running at candidate machines. Under this modality, the resource utilization would highly depend on the way pilots are distributed and the functionality they offer. Apart from the generic pilot and hierarchical pilot discussed above, another possibility is to have the pilot request multiple user jobs (hence, their payloads) simultaneously once the computing environment preparation is completed. Although such a multi-tasking pilot is theoretically possible to implement, it often leads to difficulty in maintaining fairness of resource sharing and could potentially result in machine overload. For the reasons above, achieving the optimal scheduling result is then being delegated to the pilot submission mechanism – the pilot generator.

2.3.2 Pilot Generator

In the PanDA framework, pilots are distributed to remote resources via an independent system tied to an underlying scheduler (e.g. Condor), as can be seen from both Figure 2.1 and

Figure 2.2. An advantage of this scheme, particularly for the interactive analysis in research projects where minimal latency from job submission to launch is expected, is that the pilot dispatch mechanism bypasses any latency between pilot submission and execution – the user obtains, from the remote resource, an interactive session within a short duration so long as there is at least one pilot, out of the population running over active resources, that presents a valid computing environment for the job's payload at the time of need.

In this manner, the pilot mechanism isolates workload jobs from compute resources and batch system failure modes in that a workload job is assigned if and only if the pilot successfully launches on a candidate resource. Throughput of user jobs is increased since there is a continuously ongoing service of resource provisioning from distributed pilots running in parallel as jobs are submitted. As a result, machines effectively appear available on-demand for end users. In addition, the pilot service layer isolates the PanDA system from Grid heterogeneities, being encapsulated in the pilot, such that from the perspective of end users the Grid or the resource-sharing environment in general appears homogeneous.

The PanDA framework places no restrictions on the mechanism by which pilots are disseminated. In fact, this highly depends on resource provider's preferable batch system, which is part of the heterogeneous factor of the resource-sharing landscape. The US ATLAS production, where PanDA was originally introduced, has been primarily using Condor-G to schedule pilots across site boundaries [85]. Yet, Condor-G encounters scalability issues at Globus-controlled gatekeepers as a result of high GRAM traffic in response to a high-volume of user jobs. GRAM [90] refers to the Grid Resource Allocation and Management protocol that supports the submission of remote job requests and their subsequent monitoring and control. With an increase in user job demands, often seen in complex projects (e.g. ATLAS experiment), a higher pilot flow is expected accordingly, which in turn leads to heavier GRAM traffic (i.e. the traffic within the channel between Condor and Globus software). To achieve a more scalable pilot dissemination, a distributed scheduling approach based on Condor's glidein mechanism is

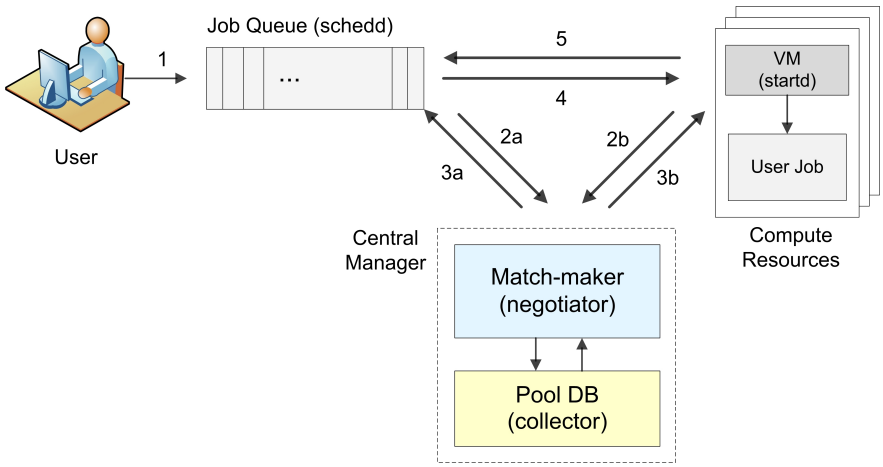


Figure 2.4 Condor kernel. The end user first submits tasks to job queue(s). Both job queues and available virtual machines then advertise themselves to the central manager, which then proceeds with an internal match-making algorithm to find eligible virtual machines for a given task.

therefore conceived. Details are to be covered in the next section following a brief overview of the Condor system.

2.4 Condor and Pilot Factory

Condor is a distributed workload management system developed primarily for integrating distributed resources to ultimately achieve both high-throughput computing [55] and opportunistic computing. Similar to other batch systems, Condor provides the following major functionality: job/machine monitoring and management, fault recovery, checkpointing, customizable scheduling polices and match-making mechanisms that reflect job/machine requirements and different priority schemes [57]. Figure 2.4 presents the fundamental structure of the Condor system. The core logical components, also known as Condor kernel [59], include job queue (functionally represented by the Condor *schedd* daemon), virtual machine (*startd* daemon), match maker (*negotiator* daemon), and in-memory database (*collector* daemon).

The following stepwise description briefly outlines how Condor works using the aforementioned components: i) users submit tasks to one or more job queues (i.e. *schedds*) in

the format of ClassAds containing matching criteria (i.e. task requirements) ii) the *schedd* publishes all task information to the pool database (*collector*) while, in the mean time, Condor virtual machines (i.e. *startds*), being distributed over all the available compute resources, also advertise their associated machine profiles (in ClassAds) including system configurations, machine runtime states, and matching preferences (over user tasks), etc to the *collector* iii) with the *collector* receiving information from both the job queue and resources, the Condor *negotiator* then executes its match-making algorithm based on the task assignment policies in terms of the requirement statement associated with both the task and the resource and finally determines the best match. Requirement statements (with the syntax similar to ClassAds) [61] for a user task could include the required platform on which to run the payload, minimum CPU speed, minimum memory capacity, and a particular demand for software stacks, etc. Similarly, requirements can also be defined for a computational resource that for instance would include the preference over particular task types and criteria to satisfy before running a user task, among many others. Interested readers are encouraged to refer to the Condor manual [61] for more examples of specifying both the task and machine requirements. Moreover, later in Chapter 7, we shall see an extended reinforcement learning framework that automatically determines the best requirement statement from the history of learning an optimal task assignment policy.

2.4.1 Condor Glidein

A basic Condor-managed resource pool consists of the following building blocks: i) the *job submitter* with one or more job queues (*schedds*) containing submitted user jobs, ii) the *job executor* consisting of one or more distributed virtual machines (*startds*) that represent all the available compute resources, and iii) the *central manager* (i.e. *collector* and *negotiator* combined) primarily responsible for collecting system-wise status information and performing the match-making algorithm. Each role mentioned above runs independently and can be

deployed on different machines. Condor system, being structurally decentralized in its design, makes it possible to dynamically deploy partial Condor functionality on-demand (i.e. a subset of Condor daemons) across the network, whereby it expands the local resource on the fly. The idea of dynamic deployment gives rise to *glidein*. A Condor glidein generally refers to the *startd* and its functionally-dependent daemons – together serving as a virtual machine – that are dynamically installed and executed on a remote resource. In particular, glidein *startd* creates an abstraction of the hosting machine in terms of the Condor representation and advertises itself to the local-pool database (*collector*) such that the remote resource effectively joins the local pool and becomes visible to the local user.

The schedd-based glidein, similar to the dynamically-deployed Condor virtual machines mentioned previously, is accomplished by remotely installing and executing a subset of Condor daemons that together function as a job queue. This remote *schedd* effectively "glides into" the local resource pool by advertising itself to the local *collector*, sharing exactly the same mechanism as the glidein *startds*.

Figure 2.5 illustrates the Condor glidein mechanism and compares the two different glidein types, with one working effectively as distributed virtual-machines and the other as a dynamic job queue. Typically, schedd glidein is deployed on a remote Globus-enabled machine where the glidein operates as a medium that redirects Condor jobs to the site's native batch system. Contrary to the job flow in the Globus model, user jobs now flow through the schedd glidein to the remote batch system rather than through the Globus Job Manager, a set of processes that perform monitoring and control over Grid jobs. The downstream flow remains the same as the Globus model where the remote batch system is responsible for eventual match-making for the incoming jobs.

The glidein schedd communicates with the native batch system through the related GAHP server process depending on the native batch system type (e.g. Condor, PBS, LSF, etc). For the purpose of presenting the glidein-based approach for pilot dispatch, here it is assumed

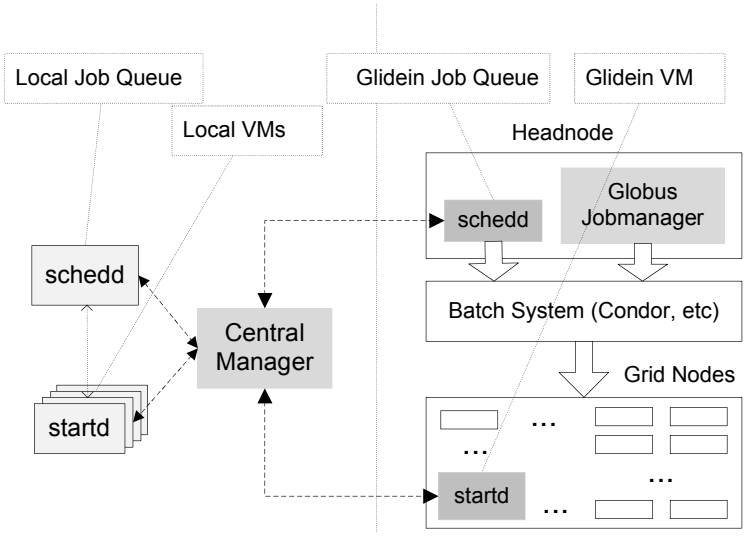


Figure 2.5 Comparisons between Startd Glidein and Schedd Glidein.

that the remote batch system type is Condor, in which case the glidein relays jobs using the Condor-C mechanism [61]. Condor-C stands for Condor to Condor, designed to interconnect two or more independent Condor-managed resource pools. Using Condor-C, jobs are submitted to a job queue (i.e. client *schedd*) and are subsequently forwarded to another job queue (i.e. server *schedd*). The client *schedd* is typically an instance running on a machine within the local resource domain (i.e. the local job submitter) whereas the server *schedd* runs in the foreign resource domain. When the server *schedd* is the glidein instance, remote jobs can effectively be allotted to a foreign computational cluster while being treated almost the same as local jobs. In this manner, multiple clusters across administrative domains are virtually merged together, thereby expanding the resource boundary.

The glidein *schedd* works similarly to the role of the Globus Job Manager in the sense that the glidein also serves as a resource broker connecting different resource domains. The fundamental difference is that the jobs dispatched via a glidein *schedd* no longer go through the Globus GRAM channel. Compared to the Condor-G using GRAM protocol, each job, either in active or wait state, is monitored and controlled by a Globus *jobmanager* process (a primary

component of the Globus Job Manager), leading to higher resource consumption upon heavier job flow. In the Condor-G model, such overhead due to job monitoring activities is ameliorated by introducing a Grid Monitor that temporarily shuts down the *jobmanager* process while the associated job is not running. However, the source of the overhead still exists due to the remaining monitoring activities in active jobs, which are required considering that each user job is unique and that the job representations are inherently different in the Condor and Globus system [63]. In particular, Condor-G converts the job description to RSL (Resource Allocation Language) format used by GRAM. The Globus Job Manger then parses the RSL that specifies the binary to be executed and other job requirements such as CPU time, number of processors, etc, some of which are further used to construct the job submit file for the native batch system.

While Condor-G fits the needs of regular user jobs, it may be excessive for the pilot mechanism since pilots in aggregate work as a light-weight service layer on top of the job payloads. The schedd glidein can thus achieve higher scalability for the pilot mechanism by treating pilot jobs as a "homogenous job stream," requiring no separate job monitoring and control. In addition, a glidein by definition is only deployed on a service-on-demand basis and thus can be removed when no jobs are intended to use the resource pool. The next section introduces an application of the schedd glidein used in pilot disseminations.

2.4.2 Pilot Factory

Pilot Factory (PF) represents an independent and automatic system for the pilot dispatch and control. It is developed in parallel with the PanDA system and for historical reasons, pilot factory took its name to differentiate it from a regular pilot generator (or pilot submitter) used only as a component in the factory. The factory first deploys glidein schedds to the head nodes of the sites, followed by the backend pilot generator submitting pilots directly to these glideins, from which these pilots are then redirected to the native batch system.

A pilot factory consists of three major components: i) a *glidein launcher*, responsible for the dynamic deployment of glideins to eligible sites ii) a *glidein monitor* that detects any failure or removal of the running glideins due to walltime limits or temporary site downtimes, upon which the monitor then invokes the glidein launcher to deploy new glidein instances and iii) a *pilot generator* that distributes pilots through the schedd glideins running on remote resources. The core of the glidein launcher and monitor lies in the mechanism to submit glidein requests, which is accomplished by initiating Condor-G jobs to configure, install and execute the related daemon set on the target head node of the foreign site. The pilot generator is built upon Condor *schedd*, to which pilots are submitted. Given that all factory components are essentially complex wrappers over Condor, they can be distributed, like the majority of Condor daemons, to different machines without locality constraints.

The current implementation of schedd glidein still relies upon the service of Globus software for its initial setup in that the glidein deployment is achieved via two consecutive Condor-G jobs: the setup and the startup. The system-setup job locates and installs platform-dependent, schedd-related binaries on the designated head node, generates the required configuration file and a startup script to be used at the next phase. The startup job then executes the script staged earlier to activate Condor daemons. Using the glidein as a job queue for pilots, Globus software only serves occasional glidein requests and thus is fully decoupled from the pilot traffic for as long as the glidein remains active.

As alluded to earlier, Condor *schedd* supports the interfacing with multiple widely-used batch systems (e.g. PBS) in addition to Condor itself with the proper configuration and the required GAHP server binary. Using the feature above, the schedd glidein mirrors the external resource domain, thereby hiding the heterogeneity of site infrastructure. The result effectively presents the pilot generator with a uniform submission portal yet without the burden of constant job control/monitoring as in Condor-G, which allows for much higher pilot flow. The pilot factory approach therefore has great potential to lift the performance bottleneck in the PanDA

architecture in light of its higher scalability and its flexibility in the on-demand deployment. The next section justifies this modality with empirical results.

2.5 Experimental Results and Analysis

Since PanDA was introduced, the US ATLAS production has been primarily using Condor-G to schedule pilots across site boundaries. At times of peak usage, large pilot traffic is often required to cope with high demand of user jobs. To alleviate the correspondingly high GRAM traffic in Condor-G-based pilot dispatch, the pilot factory approach is developed as an alternative method that substantially reduces the need of Globus brokerage by deploying Condor's schedd glideins to the front-end nodes of the remote clusters. Pilots then flow through the glideins to the native scheduling system where glideins function as tunnels that connect the pilot submitter host with the remote scheduler.

2.5.1 Resource Usage Comparison

In this section, an experiment is presented to compare the resource usage in Condor-G-based pilot dispatch with the pilot factory approach in terms of percentage CPU time and memory consumption. The experiment was conducted in a test computational cluster (OSG-ITB test bed) with 1 head node and 8 worker nodes (i.e. backend compute resources), configured with 8 job slots; that is, a maximum of 8 jobs is allowed to be in the running state on the job queue.

To ensure a continuous supply of pilot jobs, the pilot generator was configured to maintain a queue depth of 20 pilots on the submitter host so that ultimately, the 8 job slots on the remote cluster are filled most of the time; although theoretically, a persistence of approximately 8 pilot jobs (or less) should suffice. To simulate the fact that pilots in practice should remain active for as long as their associated user jobs of varying execution times, each pilot is configured to have a runtime determined by the bounded Gaussian distribution with an

Table 2.1 Pilot-specific parameters.

Parameters	Values
Number of job slots in the target cluster	8
Pilot queue depth	20
Pilot runtime	Minimum: 5 seconds Maximum: 180 seconds
Sampling rate of resource usage	15 samples per miniute

appropriate lower and upper limit (see Table 2.1). Further, the resource usage metrics (e.g. percentage CPU time) on the cluster-head node are sampled on a predefined interval perturbed by a Gaussian noise. The irregular sampling interval is incorporated here with the intent of minimizing biased measurements from any possible “synchronization” between hidden temporal patterns in the scheduling process and the sampling process itself. The specification for the experimentation is summarized in Table 2.1.

Figure 2.6 compares the resource usage in terms of percentage CPU time while Figure 2.7 compares memory usage. With the same pilot load, the result indicates that the pilot factory mode has lower resource consumption on average and lower sampling errors. Since a glidein only serves as a conduit between the submitter host and the remote scheduler, a computing resource is only allocated for running the *schedd* and the GAHP server process without additional processing time required for the per-job monitoring/control as in the case of Condor-G. Note that the experiment focuses on relatively shorter jobs with life spans within the order of a few minutes (3 minutes at maximum in this experiment). Theoretically speaking, the shorter the jobs, the higher the overhead in the case of Condor-G since each job requires a *jobmanager* process being active during the phase of job submission, file staging, and the cleanup upon job completion. Consequently, as long as shorter jobs are in the majority, the resource usage with higher workload is expected to be similar to (if not higher than) the empirical results presented

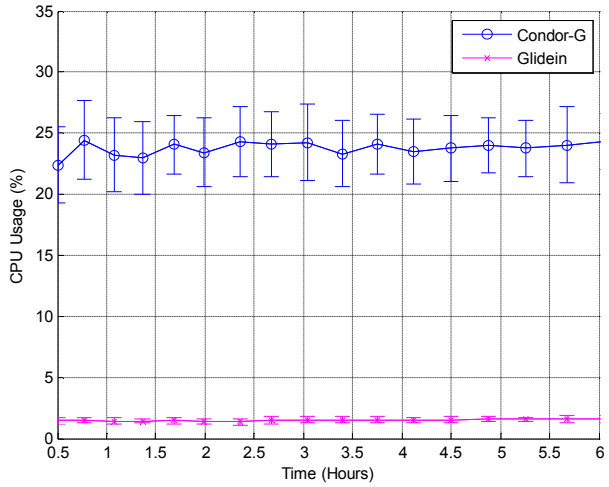


Figure 2.6 Percentage CPU times during a 6-hour time frame

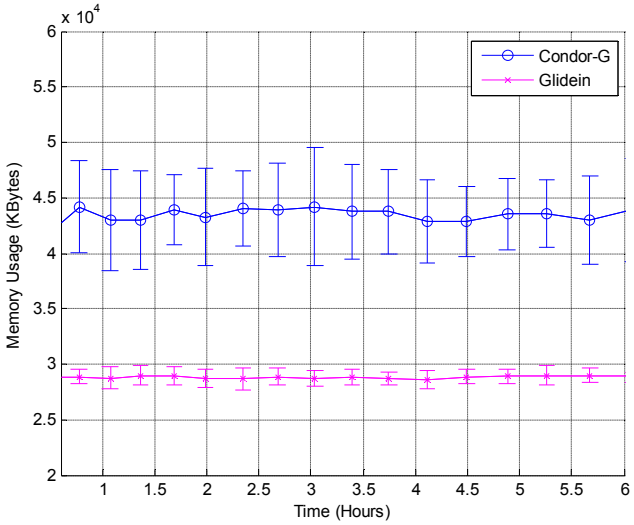


Figure 2.7 Memory usage during a 6-hour time frame.

here. Lastly, since a *jobmanager* and its child processes appear at various stages of a Condor-G job prior to its completion, both the CPU and memory usage tend to fluctuate more than those in the PF submission mode, resulting in higher sampling errors.

2.6 PanDA Experience

The development of the PanDA system dates back to late 2005 to meet ATLAS requirements for efficient processing and management of large-scaled production tasks and distributed scientific analysis. The PanDA architecture gives rise to a dynamic workload management system that optimizes resource utilization through data-driven scheduling and just-in-time resource allocation with the pilot mechanism. The benefits of PanDA's architecture have led to its widening use in OSG [78] and EGEE [76], etc. In particular, PanDA has processed more than 270 million jobs as of mid 2011, currently at a typical rate of about 1.7M jobs per week for production and about 350K jobs per day for distributed analysis at approximately 150 sites around the world. In view of the potential of combining broader forms of compute resources across geographic boundaries, the PanDA architecture is further extended to a more generic framework that adapts to the heterogeneity of Grid infrastructures, thereby providing a uniform job-management service layer and achieving optimization of resource allocations with a late-binding strategy as emphasized in this chapter. These efforts lead to a collaborative research with the Condor team and the development of applications in glidein technology such as Pilot Factory as described earlier.

2.7 Related Work

The late-binding strategy used in PanDA for resource allocation is realized in two contexts: data-driven scheduling and just-in-time match-making. In the PanDA framework, both the aforementioned services are in part delegated to the distributed pilots at the candidate resources in the sense that pilots can be configured to initiate data movements and collect real-time resource profile for match-making purposes. The concept of the pilot mechanism, in its late-binding with data, can be traced back to the experience in DIRAC [81] used for the LHCb experiment. DIRAC provides several architecture-level solutions for reliable data distribution, data integrity and access in order to minimize waste of resource due to failure modes during the

data staging process. Once the required sets of data become available and are validated, the workload agent in DIRAC then submits to the Grid the jobs that have been waiting for these data.

From the perspective of the late-binding between jobs and resources, the pilot identity is analogous to the role of remote client agents (or workers) in Volunteer Computing systems such as BOINC [51], SETI@home [80], and Distributed.net [72]. In these systems, each volunteer host is attached to the servers from which tasks can be downloaded. Various CPU scheduling schemes on the level of work-fetch policy, CPU time-slicing, estimate of completion time, etc, are then enforced by the client agent running on the volunteer host [70]. The acquired tasks, as a result, can be tightly matched with real-time machine properties of the volunteer host. Such a client-server model also exists in the form of the pilot mechanism in the PanDA architecture; yet the pilot approach works slightly differently in that there is no pre-defined agreement that associates user tasks with the target machines where pilots are injected. Consequently, the real-time resource information collected by pilot jobs is sent back to the PanDA server where the scheduling strategy is dynamically determined by selecting the best-fit user task mutually agreeable to the target resource. In this manner, scheduling algorithms are then decoupled from the client agent (i.e. the pilot) that initiates requests for user jobs.

Efficient cross-domain resource allocation is a key aspect for minimizing heterogeneity of the resource-sharing environment. This subject has been addressed by many related research projects including Condor-G, Berkeley Database Information Index (BDII) [77], and other edge services such as MDS (Monitoring and Discovery Service) from the Globus project. Condor-G now incorporates a site-level resource allocation mechanism by which user jobs are matched to the desirable sites without having to explicitly select the target site. Grid resources identify themselves by advertising their available services, requirements and preferences over jobs in the form of ClassAds while user jobs also specify the likes; a match occurs when the overall requirements between a job and a Grid resource are compatible with each other.

However, achieving this high-level brokerage requires the sites to cooperate by providing consistent and pre-defined resource descriptions that accurately reflect the capacity of their managed resources. BDII, on the other hand, periodically polls resource attributes, such as free CPUs, supported Virtual Organizations, etc, through LDAP servers gathering information from computational clusters. However, using the BDII approach for resource allocation still requires an agreement and consistency over the resource profiles from their providers. In addition, the real-time resource information is obtained through constant polling (e.g. using periodical cron jobs) to the related servers in the target site domain. This architecture, when compared to the pilot mechanism, would require dedicated servers per site and thus, may not generalize as well to the general resource-sharing environment such as the network of volunteer hosts.

2.8 Summary

Abstraction of the resource-sharing infrastructure for a homogeneous representation has been one of the primary goals in the Grid computing community. The PanDA-PF architecture is presented in this chapter to achieve a uniform view of the Grid and better resource utilization through the layer of distributed pilots and their efficient dispatch. The pilot mechanism accelerates distant resource discovery and, through late-binding between tasks and their target machines, minimizes computing resource waste in various failure modes. In the mean time, pilots gather real-time resource properties so as to make a seamless match-making procedure achievable. Heterogeneity of the Grid is encapsulated in the layer of the distributed pilots so that users do not have to deal with the differences in the underlying schedulers and other specifics in the fabric layer [64] of the Grid. Further, the PanDA-PF architecture also aims to generalize resource utilization to a broader form of distributed resources such as volunteering computing nodes across the Internet where pilots can act as client agents that initiate requests for user tasks.

Pilot Factory is a glidein-based solution to a more scalable pilot dispatch than the conventional Grid-job approach. In the pilot factory mode, the glideins as dynamic job queues are first installed at the target cluster head nodes prior to pilot dispatch. This is mainly accomplished through appropriately configured Condor-G jobs using Condor *schedd* and its related daemons as executables. Subsequently, light-weight repetitive jobs such as pilots will then flow through glideins to the native scheduler without excessive monitoring/control on the per-job basis. Treating each pilot individually as a Grid job often leads to scalability issues due to the correspondingly large GRAM traffic, which is required in pilot disseminations to cope with large and constant job flow. In particular, heavy workload is often expected during the course of large-scale and complex scientific projects such as the ATLAS experiment.

PanDA-PF WMS provides users with an architectural foundation for resource acquisition, validation and allocation with user jobs. However, there are other dimensions in harnessing distributed resources not yet fully investigated within the PanDA framework such as the following: i) strategic expansion of the resource boundary (i.e. increasing the set of available resources) by distributing Condor glideins (or VMs in general) on compute resources fitted for user jobs, and ii) adaptive match-making policy that improves itself through learning the dynamics of jobs and computing resources.

In support of the preceding objectives, glideinWMS [71] can be used to expand the resource boundary by skillfully distributing Condor glideins at the target computational clusters as locally-accessible virtual machines. Furthermore, we will further address the optimization of resource utilization and the match-making process in the subsequent chapters by developing a generalized reinforcement learning framework. This new learning framework will extend the existing RL formalism with a novel approach that enables both the learning of an action-oriented conceptual model and deriving an optimal policy simultaneously. As we shall see shortly, the concept-driven action abstraction serves as a basis for solving the match-making optimization problem characterized by morphing resource boundary formed by distributed pilots or glideins.

Methods based on automated learning mechanism could potentially increase the productivity of the PanDA-PF WMS significantly through a strategic pilot dispatch with Pilot Factory in addition to the optimal resource allocation within the PanDA framework.

CHAPTER 3

GENERALIZED REINFORCEMENT LEARNING APPROACH: A PREVIEW

We have seen in the previous chapter that in a general resource-sharing environment, the associated service hosts (e.g. Grid nodes, PCs, etc) can be characterized by hardware configurations, platforms, software stacks, and real-time resource capacity, among others. One of the primary objectives of a workload management system such as PanDA-PF is to assign user tasks to compatible computational resources in order to maximize a given performance metric (e.g. shortest turnaround time). The development of PanDA-PF sets the stage for developing an efficient task assignment strategy using a scalable pilot mechanism (i.e. Pilot Factory) that effectively discovers and reserves free computational resources as end users are submitting their tasks in parallel. The next challenge to face is to find an efficient and adaptive solution for task assignments under the set of dynamically-acquired computational resources (via pilots) that are characterized by limited walltimes with their capacity persistently changing due to the sharing of multiple users across the network.

In particular, the policy search for optimal task assignments can be formulated in terms of a sequential decision process (SDP) in which the learning agent attempts to map situations to control decisions through sequential interactions with the system, thereby gradually reaching the optimal control via successive approximations. The reinforcement learning (RL) framework [8] characterizes SDP through the state space, the action space and the system dynamics largely based on the formalism of Markov Decision Process (MDP) introduced by Bellman [91, 92]. The state space is essentially an abstraction of the control task of interest, which includes the features (or state variables) that characterize the underlying system and the learning agent. A reasonable state representation of a navigational task would be, for instance, the position of

the learning agent in addition to perhaps the constraints inherent within the agent itself such as the fuel level. The action space models a permissible set of control decisions that can be executed in the state space as the agent interacts with the system. The system dynamics capture how the state (of the system) evolves over time in response to the successive decisions made by the agent along with their corresponding system feedback. An optimal control policy thus involves an ideal action selection that induces a state trajectory leading up to a maximum payoff. An optimal navigational strategy could be to follow a relatively shorter path to the destination while taking necessary detours to avoid obstacles en route with the simultaneous consideration of energy conservation. In a similar fashion, an optimal task assignment strategy in the workload management domain would be to perform a successful match-making process such that incoming user tasks are always allotted to compatible servers with relatively higher real-time resource capacity.

While a detailed specification and formalism of RL will be deferred until Chapter 4, we shall for the moment briefly point out common technical difficulties faced by standard RL methods. In addition, through the high-level schematic in Figure 3.1, we will present the general approach proposed in this dissertation and the corresponding thought process toward lifting the limitations in the standard RL framework to achieve more adaptive policy learning and more compact representation for large engineering applications. As we shall see later in the upcoming chapters, complex control domains in general share the same learning aspects and challenges to be resolved. To arrive at a general solution applicable to a wide spectrum of control tasks, starting from Chapter 4, we will progressively introduce novel constructs and extensions to the existing RL framework in addition to the related work that had motivated the novel approach at different levels of the learning framework. This research will finally culminate in the Concept-Driven Learning Architecture (CDLA) in Chapter 7 that connects all the dots for a complete system layout referenced by Figure 3.1.

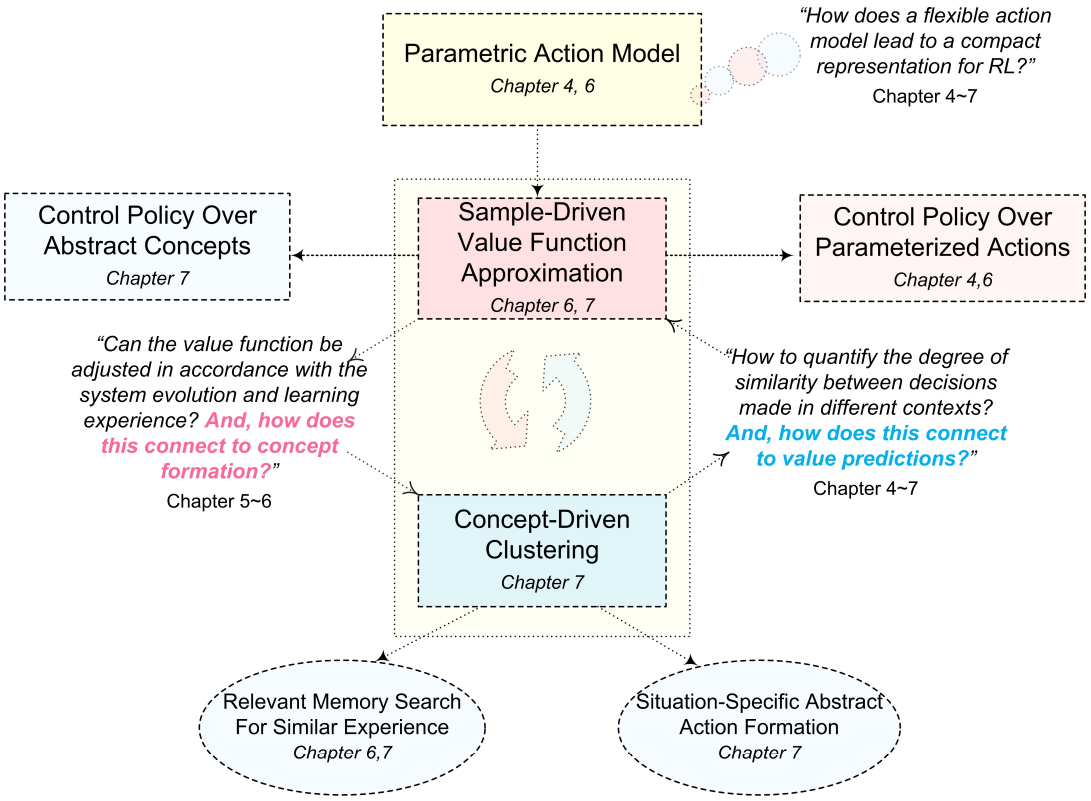


Figure 3.1 A high-level schematic of the generalized reinforcement learning framework. The objective is to enable a simultaneous learning of the control policy and abstract decision concepts. Policy learning and high-level decision concept formation mutually leverage the strength of each other. The key is to identify missing links among three components: (1) value function approximation (2) experience association that relates similar decisions and (3) abstract concept formation. Additionally, a periodic updating mechanism is required to progressively adjust the learned control policy in response to environmental shifts.

3.1 Overview of the Reinforcement Learning Approach

There are four fundamental building blocks in reinforcement learning systems: a control policy, a reward function (or cost function), a potential function (or value function), and optionally, a model of the environment. A control policy specifies the action to take in a given state of the system, which could be as simple as a table lookup or an extensive computation involving a complex search process. A reward function maps each perceived state to a real value indicating an intrinsic desirability of being in that state. This is effectively how the system

provides its immediate feedback to the learning agent in response to the action taken in the state. In contrast to the immediate system response as a reward or cost, accumulated reward in the long run (sometimes referred to as the delayed reward) is often what determines a good decision in a given situation. This is evaluated through the potential function that takes into account the system dynamics such that the value of actions applied in associated states will reflect the long-term desirability. For instance, an action may yield a very high immediate reward in a state but nonetheless still end up with a relatively low potential value since this state is more often than not followed by other states with low rewards (or high costs).

The last key component of some reinforcement learning methods is a model of the system, which could be anything that enables the agent to predict how the system will respond to a course of actions without the need to experience them. For instance, in Chapter 4, we will briefly cover the dynamic programming (DP) approach in which the model corresponds to estimates of state transition probabilities and expected rewards. A system model can usually be categorized into two major types: i) distribution models and ii) sample models [8]. A distribution model generates all possible situations in the decision process (e.g. possible sequences of state-action pairs) and their corresponding probabilities to occur. On the other hand, a sample model generates one possible scenario at a time according to the probability distribution that underlies state transitions and system responses to actions. This dissertation will introduce a third category of system model based on experience association that predicts the consequence of a control decision in terms of its degree of correlation with other related past experiences (see also Figure 3.1). In essence, experience association is inspired by the psychological mechanism of searching relevant memory from the past that is similar to the current situation where a decision is to be made. Taking the task assignment problem for example, if a particular type of user task was processed successfully on a certain group of computational resources, then it is very likely that one can reproduce the same preferable result by matching new tasks with similar attributes (e.g. similar demand in memory) to those resources with similar resource

capacity. Experience association can be implemented based on sampling or a probabilistic model but more importantly, a reasonable similarity measure is required to define the meaning of similar decisions made in different contexts.

3.2. Challenges in Reinforcement Learning Systems

Large, complex and varying environments present to the stand RL framework with several challenges to resolve, including i) compact representations for large and continuous state/action spaces ii) adaptive policy learning in response to a changing environment iii) abstraction over the state and the action space for policy generalization and reuse under similar situations. The generalized reinforcement learning approach to be established in the subsequent chapters aims to provide a clean solution that unifies these learning aspects into a coherent learning framework. First, we shall briefly describe these challenges, followed by the direction we are aiming in to solve them in Section 3.3.

3.2.1 Challenges from Large State and Action Spaces

The advancement of reinforcement learning systems has led from solving limited problem domains with discrete states and actions to complex, realistic domains involving continuous state and action spaces [6, 113]. However, most of the contemporary RL methods often assume that actions are merely an atomic and prefixed set of control decisions, be it discrete or continuous. Under such a formulation, actions collectively serve as a mechanism for triggering state transitions. As a consequence, the complexity of RL methods depends largely on the dimensionality of the state space representation, which in turn governs how effectively the (potential) value of a particular decision can be evaluated. Challenges arise when applying RL methods to real-world control tasks that require numerous features to be accurately represented, leading to an exponential growth in the state space and difficulty in representing

the control policy. This is what Bellman referred to as “the curse of dimensionality” since the computational requirement increases exponentially in the number of state variables.

3.2.2 Challenges from Irreducible and Varying Action Sets

The decision process can be further complicated by the presence of a relatively large set of target actions, which are globally irreducible since they are all significant for the underlying task. This is the case in the task assignment problem induced by PanDA-PF WMS in that the distributed pilots can in principle discover a large set of computational resources in parallel to user task submissions. In addition, in a resource-sharing environment, the availability of computational resources at different times may not remain consistent due to the resource sharing among multiple users across administrative domains. The desired task assignment policy thus needs to simultaneously consider a potentially large and varying set of pilot-resident hosts, from which to find the most compatible subset to optimize a given performance metric.

3.2.3 Challenges from Environmental Shifts

In addition to learning prone to the abovementioned complexity problem, standard RL often lacks also the ability to efficiently adapt to an evolving environment or unexpected behaviors from the action. These two seemingly different issues are indeed closely related. Consider a robotic navigation domain, for instance, where the varying environment dynamics could be related to a newly-formed rough surface due to bad weather or a misaligned motor caused by inevitable damage from long-term use. These unpredictable environment shifts, including variations from within the agent itself, can at any moment drive the agent away from the course necessary for the learned policy to unfold toward a maximum payoff in that the policy may no longer be aligned with the current environment. Under such conditions, the learning agent is effectively experiencing non-stationary action outcomes since the implication of taking the very same action does not continue to apply in the future. This is also the case for task

assignments in a resource-sharing environment in which the task scheduler perceives constant fluctuations in the real-time resource profile of candidate machines. An optimal task assignment policy learned earlier will certainly need to be adjusted accordingly in order to fit the varying resource dynamics. However, such real-time policy adjustments from the standpoint of standard RL methods often equates to a complete re-learning process from the start in that the varying environment effectively redefines the underlying RL problem.

3.3 The Puzzle of Integrating Function Approximation with Abstract Concept Learning

Function approximation methods [2-5] have been used extensively to address the dimensionality barrier in complex systems by using a set of basis functions to represent the value function. In this manner, the value prediction need not be constrained as greatly by the size of the state and action space. The generalization capability of the function approximator enables reasonable value predictions in a larger and possibly uncharted state space area by using only the experiences from a smaller, limited subset of the state space. However, care must be taken in choosing an appropriate size of the basis set in addition to their functional forms that generalize well enough to reflect the geometric property of the large state space. Also, the size of the basis set can grow rapidly with the dimension of the state space.

Forming a high-level abstraction over the state space based on temporal structures, subgoal discovery, hierarchical policy learning, common substructures from within the state space representation, or policy representation, etc, is another thread of research in scaling reinforcement learning systems. For instance, the notion of the variable temporal resolution model (VTRM) is established based on the observation that interesting events that contribute to value prediction occur at various frequencies and thus different temporal resolutions should be applied in different parts of the state space for value updates. H-DYNA [118] constructs temporal abstractions in terms of hierarchical VTRMs via learning multiple tasks simultaneously, which in turn speeds up the learning for subsequent tasks. The notion of VTRM eventually

evolves into the concept and formalism of Semi-Markov Decision Process (SMDP) with the *option framework* [34], where options are sometimes referred to as *skills* [120] depending on the context. Temporal abstractions in SMDP accelerate policy learning by taking "larger steps" toward the goal in units of options where each option expands to a course of actions taken from a state and has its own objective and a corresponding termination condition. Factored MDPs [96, 97], on the other hand, reduce the representational size of the state space by expressing states implicitly through a set of random variables and subsequently make use of the structural regularity within the state transitions in response to the stochastic effect of actions. Similar in a way but with inherently different representation, the SMDP homomorphism framework [35] builds an equivalent yet reduced MDP out of the structural symmetry and redundancy in the state space. The key idea of policy-oriented, context-dependent structural symmetry across different regions of the state space (or the state spaces across multiple similar tasks) gives rise to the framework of state abstractions, which eventually evolves into a line of methods that allow for skill chaining [120], skill reuse or transfer [119] across similar tasks. Hierarchical Reinforcement Learning (HRL) [18, 19] decomposes complex control tasks with larger state spaces into different smaller subtasks (or subgoals) in which individual control policies are learned. In particular, HRL in some sense can be thought of as a unifying framework for task decompositions and hierarchical policy learning based on various schemes of abstractions including the temporal and state abstractions mentioned previously.

Although both of the methodologies above – function approximation and abstract concept learning – can be shown to scale standard RL to larger and more complex environment, however, they address the RL challenges mentioned earlier in Section 3.2 from two separate directions rather than leveraging the strength of each other to form an even more powerful learning approach. Note that in HRL, due to the mechanism of task decomposition, the computational cost is indeed reduced in learning individual value functions for the local policy within each subtask. However, the value function decomposition in this sense is subject to the

particular definition of concept hierarchy which in many cases would require programmer-designed heuristics or predetermined hierarchical structure such as the MAXQ graph [21]. Even within the framework of automatic subgoal discovery [19, 34], there is no guarantee that a sensible subgoal area will always stay sufficiently small in order for value estimates to become efficient. Further, with the presence of high-level value functions and coarse-grained value functions defined for subsets of state partitions, the parameters within the entire set of value function estimators could still become significantly large for complex systems, leading to difficulty in providing convergence guarantee.

Similarly, deriving an adaptive control policy that copes with nonstationary action outcomes with a large, irreducible and varying action set is an inherently difficult problem due to the limitations within the standard RL formulation as we shall see shortly in later chapters. Integrating the value function approximator with the learning of high-level abstractions would require an intermediate structure that bridges the differences in their representations.

3.3.1 An Outline of the New Paradigm

Starting form these challenges, this dissertation sets out to develop an RL approach that combines an adaptive value function and policy search, state abstraction and action abstraction learning in the context of a parametric action model. It was not straightforward, when viewed from the initial phase of this research, to identify an appropriate bridging representation that connects function approximation and concept formation together such that both types of learning can take place at the same time without compromising their separate performances. An initial sketch of the notion for realizing such an interleaving process is depicted in Figure 3.1 in terms of the center box (in a soft, blurry yellow), where we have shaded the coloring and also raised the questions to be answered in the later chapters as a setup for a more complete and detailed schematic when new concepts are introduced.

The functionality and interactive components in Figure 3.1 took their shape also from the fact that the value function needs to be updated in response to environment shifts or varying action outcomes. At the same time an update in the representation of the value approximator also needs to efficiently propagate to the conceptual model such that the following cycles of policy learning (over these conceptual units) will continue to reflect the current environment dynamics.

Given the challenges of real-world systems, it is our postulate that a more generalized expression for actions and their pairing with states will lead to a more compact and scalable RL representation and also further facilitate the notion of context-dependent policy abstraction. The model developed here is motivated by the observation that complex actions in real-world applications often involve a variational procedure as they are performed. For example, when a robotic arm fetches an object at a nearby location, several parameters expressing joint movements are varied simultaneously and continuously. Similarly, when a user task is assigned to a candidate host for processing, the resource profile continues to fluctuate in a stochastic manner. As such, the execution of an action in practice naturally involves an intricate and continual interaction with the environment, which we will refer to as *action dynamics* for reasons that will become clear later in Chapter 6. Similar to the state abstraction with features that represent the key aspects of the environment, complex actions can be mathematically modeled by a set of parameters that approximate their localized influence over the external environment as they are performed. As we will begin to see in Chapter 4, the new action model described here serves as a seeding mechanism for the other components of the generalized reinforcement learning framework to fall into place in a manner supported by a rigorous mathematical framework from functional analysis [104]; in particular, the notion of kernel methods [1, 22]. To develop this framework, we will begin, in the next chapter, a systematic presentation of the generalized RL approach with the parametric action model represented by the “head” of the robot-like schematic in Figure 3.1. We will then introduce a kernel-based

function approximation framework in Chapter 5, leading to the concept of a Reinforcement Field in Chapter 6. Finally, Chapter 7 will develop an approach to decision concept formation and integrate all the components, including value function approximation along with its dynamic adjustments, relevant memory search, and adaptive policy learning over high-level conceptual units, into a complete learning architecture. Metaphorically speaking, the conceptual robot depicted in Figure 3.1 will finally pump the heartbeat, which symbolizes the periodic, interactive processes between the value function approximator and abstract concept learning.

CHAPTER 4

REINFORCEMENT LEARNING AND PARAMETRIC ACTION MODEL

We now embark on the formalism of the reinforcement learning framework with a focus on temporal difference learning techniques that will serve as the foundation for implementing the control learner that supports other learning elements to be subsequently developed in the later chapters. In particular, a heuristic action model will be introduced in this chapter as a partial solution to the learning architecture depicted earlier in Figure 3.1. As a roadmap toward a complete learning architecture, Figure 4.1 emphasizes the key component addressed by this chapter in colors while shading the other parts of the system in gray. Later in this chapter, we will also introduce a cluster-based action model that aggregates sets of similar actions, based on which the control policy is derived. While the action abstraction involves a few strong assumptions for the moment, it serves as a useful technique in compressing the action space involving an irreducible action set that, for instance, occurs in the task assignment domain described earlier in Chapter 2.

As a brief review and a setup for an automated learning solution to the task assignment problem, consider the following scenario in a PanDA-PF where users are assigning their tasks to a morphing metagrid (Section 2.2): a continuous stream of user tasks of various types are waiting to be allocated to a joint resource pool in which Grid nodes are contributed by different Virtual Organizations (VOs). A grid resource manager dynamically scavenges the computational power from some number of Grid nodes with sufficient resource capacity and presents them as virtual machines (VMs) to the end users across administrative domains. In this manner, users perceive the Grid as an abstraction with dynamically-deployed clusters, or a metagrid, where new VMs may continue to join the pool while other VMs exit upon reaching their walltime limits.

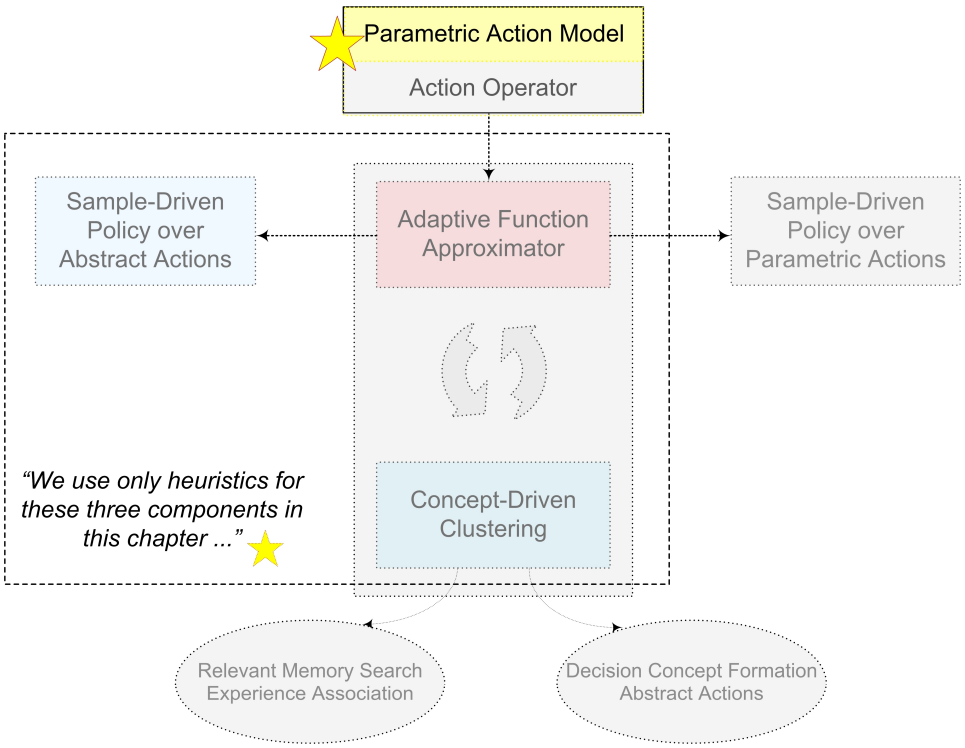


Figure 4.1 Parametric action model in the generalized RL framework. Chapter 4 will introduce the first parametric action model along with an abstract policy learning algorithm over clusters of similar actions. The component marked by a star represents the primary focus of the chapter. Components with shaded colors (and fonts) correspond to partial solutions. Gray areas are to be covered only in later chapters.

While the set of all possible allocations to the VMs is significantly large, the free VMs with a nonzero job slot at any time may vary depending on the overall workload. Moreover, since the computational resource is ultimately shared, the machine capacity varies over time on top of the already diversified hardware configurations and software stacks. Given the scenario above, among all the possible resource allocations, identifying the factors that determine a good match between a set of user tasks and a set of VMs then becomes crucial for an ideal task assignment policy. Note that task assignment and resource allocation are considered as flip sides of the same coin in this dissertation. From the perspective of standard reinforcement learning (RL), for instance, the set of all possible resource allocations corresponds to an action set with each control decision corresponding to a particular binding between a user task and a target server.

This is how control decisions can be defined for an intelligent agent in a planning or control domain. The agent perceives the world in terms of an abstraction represented by states, each of which is characterized by the features relevant to the decision process. In the scenario given above, the decision process lies in the task assignment policy. Thus, the primary goal of a reinforcement learning procedure is to identify in real-time the appropriate hosts matching the user tasks in order to optimize a given performance metric (e.g. minimized turnaround time, etc) after some experience of trials and errors. In this chapter, we will first introduce the theoretical basis for the RL framework. Next, we will point out the challenges faced by standard RL methods in complex domains such as the task assignment problem with volatile, non-persistent Grid resources mentioned above. Subsequently, we will introduce a new extension to the existing RL framework using a *parametric-action model*. The extended RL framework developed in the chapter can be considered as a preliminary step toward a full-fledged RL-based solution to efficient task assignments as well as other complex control domains in general.

4.1 Reinforcement Learning

Reinforcement learning [8] has been typically used in various dynamic control tasks such as developing strategies for elevator dispatching [102], channel allocations in cellular phone system [101], etc, and has also been gradually adapted to an even wider range of advanced and complex domains such as distributed database queries, autonomous robotic applications in interplanetary rover [103], and deep-ocean exploration, etc. Current reinforcement learning methods are largely formulated under the Markov assumption as a Markov Decision Process (MDP) or as its extension, Partially Observable Markov Decision Process (POMDP) to address further issues in the uncertainty of states due to partial accessibility to the environment at work. This section first illustrates the reinforcement learning framework in which the agent operates in a MDP.

Prior to the formal definition, first consider the agent interacting with a given environment (or a system) by making a succession of decisions. The environment is modeled by some features whose values jointly define a state *s*. The state serves as an abstraction of the environment and is the sole information available to the agent. In each state, the agent has the option of taking one of the actions available as a control decision by trading off their values (or benefits), and subsequently executes the action of choice and receives a feedback signal from the environment. The agent then moves on to the next state where the same one-step decision process is repeated over and over such that the traversed states unfold as a sequence with corresponding decisions. As an example for visualizing the decision process, Figure 4.2 depicts a navigation domain in which the rover is attempting to gather certain mineral resources within the shortest time frame while avoiding craters and other obstacles along the way. In this scenario, a state can be represented by the rover's current coordinate, the remaining energy and possibly other features relevant to the goal of interest. If the goal is to collect a certain quantity of the resource, then the amount of resource gathered could be part of the feature set that models a state. For simplicity, assume that all of the above process occurs in a series of discrete time steps $t \in \{0,1,2,...T \mid T < \infty\}$. Taking a closer look at the decision process, one can identify a few key components: a set of state representing a particular configuration of the environment, actions available in a given state, environment feedbacks as rewards (or costs), and a one-step dynamics of the environment that characterizes how a state transitions into its successor state in response to the action taken. Note that the internal state of the agent is considered as part of the environment (e.g. the remaining energy in the navigating rover). The key to the Markov property is this one-step dynamics. If the system reacts to the action taken at time *t*, stochastically in a manner that depends only on the current state and the action taken but not the past history beyond this current step, then the dependence between the system and the agent's decisions is Markovian and thus, satisfies the Markov property. More formally, for

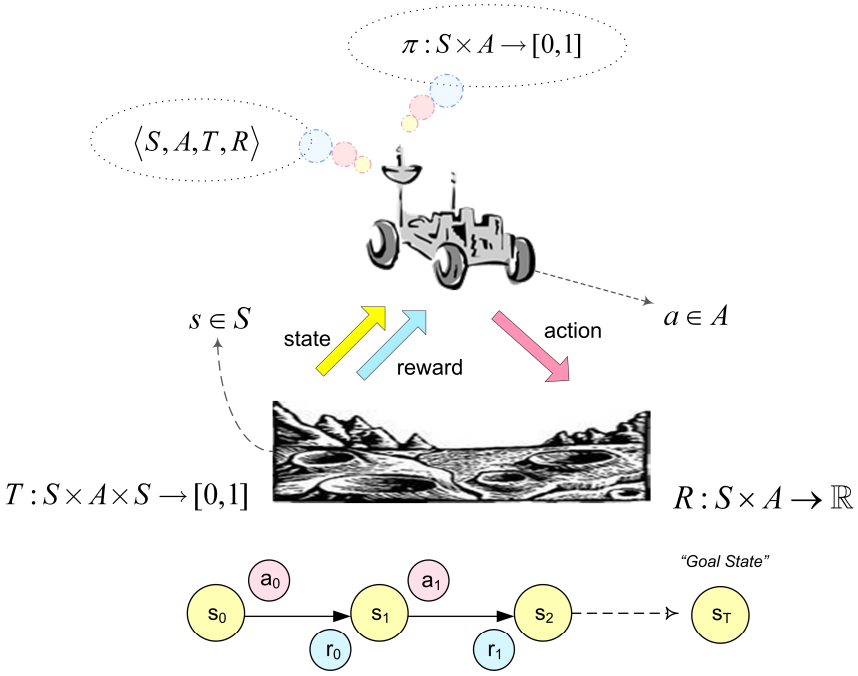


Figure 4.2 Markov Decision Process. A rover navigates on a planetary surface following an MDP in which the transition function *T* and reward function *R* capture the environment dynamics in terms of feedback signals for the rover to update her internal view of the world.

any state *s* and action *a*, the probability of moving into the next possible state *s* can be expressed by the following equality:

$$P(s_{t+1} = s', r_{t+1} = r \mid s_t, a_t, r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0)$$

$$= P(s_{t+1} = s', r_{t+1} = r \mid s_t, a_t) \tag{4.1}$$

Note the independence of the sequence $s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$ in the conditional probability measure $P(s_{t+1} = s', r_{t+1} = r \mid s_t, a_t)$. In other words, the probability of moving from s_t to the successor state s_{t+1} does not depend on the past events $r_t, s_{t-1}, a_{t-1}, \ldots, r_1, s_0, a_0$ but only on the current event (or a decision point) expressed as a state-action pair (s_t, a_t); meanwhile, note that r_t is the reward obtained in response to the past event (s_{t-1}, a_{t-1}) and thus does not

appear as a dependent random variable in (4.1). Quantities evaluated in this manner are called transition probabilities. We now formalize the Markov Decision Process in the next section.

4.1.1 Markov Decision Process

A decision-based learning task that operates under the assumption of the Markov property is called a Markov Decision Process, or MDP. An MDP is defined by the tuple $\langle S, A, T, R \rangle$ where S is a set of states, A is a set of actions, T represents the environment dynamics and R specifies an expected reward (or cost) in response to an action taken in a state. In particular, $T : S \times A \times S \to \mathbb{R}$ is a transition function that maps to a probability measure upon observing s' by executing action a in s. Equivalently, T can be interpreted as a Markovian transition operator that acts on a particular state-action pair (s,a) and returns a probability distribution over the next state $s' \in S$ in terms of the set $\left\{ P_{ss'}^{a} \mid s' \in S \right\}$ following the evaluation of (4.1). Note that the resulting probability $P_{ss'}^{a}$ corresponds to the expression $P(s' \mid s,a)$ given earlier (see (4.1)), where we have omitted the time indices to simplify the notion. On the other hand, during the state transition the agent receives a reward (or a cost) governed by the reward function $R : S \times A \to \mathbb{R}$ with its range depending on the starting state s and the corresponding action a. In the most general settings, however, the reward function R can also take into account of the next state to give: $R : S \times A \times S \to \mathbb{R}$ depending on the system of interest.

A decision process based on the MDP is to consider each state as having a value defined as a measure of how good it is to be in that particular state. Formally, this is to define a mapping $V : S \to \mathbb{R}$ where V takes on a state and returns a real value. Alternatively, a vector (possibly of infinite dimension) $\mathbb{R}^{|S|}$ can be defined through V evaluated at the corresponding sequence of states. The objective of the agent is to learn a control policy $\pi : S \times A \to [0,1]$ that

leads to a maximum accumulated reward, where π effectively maps a given state $s \in S$ to a probability distribution over actions. Thus, given a policy π, the corresponding value function $V^{\pi}(s)$ specifies the long-term accumulated reward by starting from the state s and following the policy π thereafter. Formally, $V^{\pi}(s)$ is defined as:

$$V^{\pi}(s) = E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \mid s_t = s\right], \tag{4.2}$$

where $E_{\pi}[\cdot]$ denotes the expected reward received by the agent following the policy π. Notice that an additional discount factor γ is introduced to each reward received beyond the current state s_t so that the accumulated reward evaluated through $E_{\pi}[\cdot]$ will eventually converge. This formalism also coincides with the intuition that the future feedbacks from the environment (in terms of the subsequent rewards) should have relatively less impact on the current decision compared to the current feedback. By taking note of the iterative structure within $E_{\pi}[\cdot]$, (4.2) can be further expressed in the form of the Bellman equation [8, 92]:

$$\begin{aligned}
V^{\pi}(s) &= E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \mid s_t = s\right] \\
&= E_{\pi}\left\{r_{t+1} + \gamma \sum_{k=0}^{\infty} \gamma^{k} r_{t+k+2} \mid s_t = s\right\} \\
&= E_{\pi}\left[r_{t+1}\right] + E_{\pi}\left\{E_{\pi}\left[\gamma \sum_{k=0}^{\infty} \gamma^{k} r_{t+k+2} \mid s_t = s\right]\right\} \\
&= E_{\pi}\left\{r_{t+1} + E_{\pi}\left[\gamma \sum_{k=0}^{\infty} \gamma^{k} r_{t+k+2} \mid s_t = s\right]\right\} \\
&= \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P(s' \mid s,a)\left\{R(s,a) + \gamma E_{\pi}\left[\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+2} \mid s_t = s\right]\right\} \\
&= \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P(s' \mid s,a)\left[R(s,a) + \gamma V^{\pi}(s')\right]
\end{aligned} \tag{4.3}$$

In the last two equalities in (4.3), we have used r_{t+1} to represent an immediate reward received in the state $s_{t+1} = s' \in S$, and $R(s,a)$ represents such a reward explicitly as a result of taking action $a_t = a$ in the state $s_t = s$ but without time indices to keep the notation uncluttered; i.e. $R(s,a) = R(s_t, a_t) = r_{t+1}$. Note that $R(s,a)$ can be redefined as an averaged reward with respect to the policy π and the state transition probability $P(s' \mid s,a)$ to be more consistent with the MDP formulation given earlier. Doing so allows for the term $R(s,a)$ to be pulled out from the expectation and thus leads to the following equivalent expression for (4.3):

$$V^{\pi}(s) = R(s,a) + \gamma \sum_{a \in A} \pi(s,a) \sum_{s' \in S} P(s' \mid s,a) V^{\pi}(s'). \tag{4.4}$$

With the iterative definition of $V^{\pi}(s)$ given in (4.3), the optimal policy π^* can thus be evaluated by solving the corresponding optimal value function $V^*(s)$; that is,

$$V^*(s) = \max_{a \in A} \sum_{s' \in S} P(s' \mid s,a) \left[R(s,a) + \gamma V^{\pi}(s') \right] \tag{4.5}$$

In order to access quantitatively a particular decision made in a state $s \in S$ by incorporating the corresponding action taken, the value function map $V : S \to \mathbb{R}$ is extended to $Q : S \times A \to \mathbb{R}$, where Q returns the utility (or Q-value) for the agent to take action $a \in A$ in a state $s \in S$. In most occasions, the value function $V(s)$ can be thought of as an expected value of $Q(s,a)$ over all possible actions that can ever be taken at a given state $s \in S$ provided that such expectation exists. That is, given a policy π, $V^{\pi}(s) = \sum_{a \in A} \pi(s,a) Q^{\pi}(s,a)$ if the summation exists and sums to a finite value (which would require that $Q^{\pi}(s,a) < \infty$). Similar to the derivation in (4.3), $Q^{\pi}(s,a)$ can also be expressed in the form of the Bellman equation:

$$Q^{\pi}(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s' \mid s,a) \sum_{a' \in A} \pi(s',a') Q^{\pi}(s',a'). \tag{4.6}$$

In (4.6), Q^{π} evaluates the expected value to be received by starting in state *s*, taking action *a* and following π thereafter. For this reason, Q^{π} is also referred to as an action-value function following π. An optimal policy π^* thus is associated with the optimal action-value function:

$$Q^*(s,a) = \sum_{s' \in S} P(s' \mid s,a) \left[R(s,a) + \gamma \max_{a' \in A} Q^*(s',a') \right] \tag{4.7}$$

In other words, the optimal policy π^* will always return the action that leads to maximum Q-value at any given state in the averaged sense. Solving (4.3) or (4.6) iteratively for all states gives rise to the framework of dynamic programming (DP). However, DP requires a complete knowledge of the environment dynamics *T* (so that $P(s' \mid s,a)$ can be evaluated) and reward function *R* prior to the learning process, which may not be always feasible. For instance, in a complex domain such as the task assignment in a metagrid given at the beginning of the chapter, the state transition probability depends not only on the task but also on many other time-varying factors such as the set of available target machines and their real-time resource capacity. Thus, the exact form of a complete transition probability distribution is not attainable but has to be estimated. This is where the framework of the reinforcement learning comes into play. Monte Carlo methods are the examples where samples of historical sequences involving the state-action pairs and their corresponding rewards are collected to form sample episodes [8]. These sample episodes collectively serve as the model for the environment, from which value functions and corresponding policies are iteratively approximated. This dissertation will use temporal-difference learning that combines both the advantage of DP and Monte Carlo methods to develop our first extension to the reinforcement learning framework.

4.1.2 Temporal Difference Learning

Temporal difference (TD) methods, similar to Monte Carlo methods, can learn directly from raw experience without a model of the environment dynamics (i.e. T and R). Yet, similar to DP, TD methods update utility estimates (e.g. $V^{\pi}(s)$) based in part on other learned estimates. Specifically, recall from (4.3) that the value of a state is expressed iteratively as an expected value of the immediate reward combined with a value estimate of the next state:

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+1} \mid s_t = s\right] \\ &= E_{\pi}\left\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^{k} r_{t+k+2} \mid s_t = s\right\} \\ &= E_{\pi}\left\{r_{t+1} + \gamma V^{\pi}(s_{t+1}) \mid s_t = s\right\} \end{aligned} \tag{4.8}$$

TD learning first samples from the probability distribution that defines the current policy (for current state s) and uses the resulting estimate V_t instead of the true V^{π} to obtain progressively better estimate for V^{π}. This results in the following basic form of the update rule in TD methods:

$$\begin{aligned} V_{t+1}(s) &= V_t(s) + \alpha\left[r + \gamma V_t(s') - V_t(s)\right] \\ &= (1-\alpha)V_t(s) + \alpha\left[r + V_t(s')\right], \end{aligned} \tag{4.9}$$

where $\alpha \in [0,1]$ represents a learning rate or step-size parameter that usually takes on a small number. In other words, the new estimate for $V^{\pi}(s)$ at the iteration $t+1$, i.e. $V_{t+1}(s)$, is taken to be a weighted average of the older estimate and a proxy estimate of the value s into the future that holds a new piece of information obtained the environment; i.e. the term $\left[r + V_t(s')\right]$ with an immediate reward r. This dissertation will put special focus on two popular TD-based reinforcement learning algorithms: i) On-policy TD control with SARSA ii) Off-policy TD control with Q-learning. We shall see shortly more generalized RL framework that extends upon these

1: Initialize $Q(s,a)$ arbitrarily for all state action pairs.
2: **Repeat** for each episode:
3: Initialize s.
4: Choose a from s using the policy derived from Q (e.g. ε-greedy)
5: **Repeat** for each step of the episode:
6: Take action a, observe r, s'
7: Choose a' from s' using policy derived from Q (e.g. ε-greedy)
8: $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
9: $s \leftarrow s'$; $a \leftarrow a'$
10: **Until** s is terminal

Figure 4.3 SARSA [8].

two TD learning methods in the later discussion although the generalization can also carry over to other TD methods.

Similar to the idea of the one-step proxy estimate given above, SARSA uses $\left[r + \gamma Q(s',a')\right]$ as a one-step proxy estimate for $Q^{\pi}(s,a)$. This results in the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right] \tag{4.10}$$

Figure 4.3 summarizes the SARSA algorithm in a procedural form. In particular, SARSA is an on-policy TD control method because the behavioral policy is the same as the estimated policy; that is, the agent will follow exactly the policy defined by the current estimate of $Q(s,a)$. This can be seen from line 4, 7 and 8 in Figure 4.3 where the agent under SARSA will resolve an action at a given state according to the current policy (that defines the preferable behavior for the moment) and subsequently incorporate new information (in terms of the one-step TD) using the update rule in line 8 to define the improved version of the policy (as a new estimation). Thus, the policy used for control coincides with the one being estimated. On the other hand, Q-learning [8, 9] is an example of an off-policy TD control algorithm because the policy under estimation is not the same as the one being followed by the agent. Q-learning with one-step proxy estimate (i.e. one-step Q-learning) has the following update rule:

1: Initialize $Q(s,a)$ arbitrarily for all state-action pairs.
2: **Repeat** for each episode:
3: Initialize s.
4: **Repeat** for each step of the episode:
5: Choose a from s using policy derived from Q (e.g. ε-greedy)
6: Execute action a, observe r, s'
7: Choose a' from s' using policy derived from Q (e.g. ε-greedy)
8: $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$
9: $s \leftarrow s'$
10: **Until** s is terminal

Figure 4.4 Q-learning [8].

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \tag{4.11}$$

With (4.11), the learned action-value function Q directly approximates Q^*, the optimal action-value function, in a sequence that eventually gets arbitrarily close to $Q^*(s,a)$ for any given state-action pair (s,a). The convergence of Q-learning is guaranteed as long as each state-action pair continues to be updated infinitely often and the rewards are bounded within a certain range. Figure 4.4 summarizes the Q-learning algorithm. Comparing line 5, 7 and the maximum operator in line 8 from Figure 4.4 will reinforce the idea that Q-learning is an off-policy learning method – the policy used for control is not the same as the one being estimated.

With the definition of the Q-learning algorithm given, we will further explore in this chapter our first extension to the RL framework using Q-learning as the baseline learning algorithm applied to a more generalized action model. In Chapter 6 and 7, we will subsequently introduce a more generalized learning framework and will revisit SARSA as an example to demonstrate the new framework.

4.2 The Approach toward a Generalized RL Framework

The first challenge that arises in a complex domain such as the task assignment domain is the presence of a potentially large and varying population of available actions. In metagrid scheduling, this equates to a large and varying set of possible resource allocations to the VMs available at any given time. Also note that these choices of resource allocations are irreducible, meaning that they all need to be considered as possible action targets from the perspective of a decision process. For a RL algorithm, a large, irreducible set of actions can lead to an excessively large search space and cause computational overhead during policy learning. The learning algorithm developed in the following sections attempts to reduce the action space, or rather, compress the action space by grouping similar actions that are likely leading to very similar future results. The state space could also be compressed along with the action space provided that the features in both spaces are correlated. Actions are considered similar if they, with high probability, lead to future results with sufficient commonality. Thus, the first step to the extension of the RL framework is to develop the following cluster-based action conceptual model based on the MDP formalism: if a set of actions at any given state tends to yield similar accumulated rewards based on the past learning experience, then they are very likely to reproduce the same results in the future and hence can be grouped together and considered as a whole at the point of decisions.

In a nutshell, we start with a given reinforcement learning method as the control learner and adopt a few assumptions at first to simplify the required action model from which to derive the policy. Different from standard reinforcement learning algorithms, the ultimate goal of a generalized RL framework is to derive the policy at the level of action abstractions, each of which essentially conceptualizes similar decisions in terms of both actions and their *decision contexts* (i.e. related states). In other words, we wish to equip the agent with a knowledge representation that indicates the reason that certain actions are preferable under certain situations and subsequently use the abstraction as a unit of control decision (instead of the

primitive or “raw” actions). The conceptual model learning is performed in parallel to the optimization process of the underlying policy. This is how the agent can learn to generalize its behavioral policy to unknown yet similar situations. To facilitate the formulation of abstract actions, we first introduce the concept of action parameterization, which is to be compared with the existing formalism under MDP where a fixed set of “concrete” actions is used for the control. Parametric actions then serve as the foundation for the RL agent to take a step further by associating similar experiences that occurred in the past. Organizing the past experience in this manner enables the agent to take advantage of the good decisions made earlier in similar contexts while it avoids repeating the same mistake again.

4.3 Parametric Action Model

Actions in real-world scenarios are often quite complex and beyond what a simple enumeration of available options can express. In addition, as an action is taken one step at a time, it also affects the local environment properties. In high-precision control tasks such as vehicle steering or robotic control, for example, more than a few physical attributes may be required to describe a particular movement (as an action) such as steering the vehicle by a certain speed with a particular angle, or using a robotic arm to collect a nearby object with a stretching movement characterized by shifts in 3D coordinates, etc. Upon the action taken, side effects such as removal of objects may also be introduced locally to the environment where the action took place. In a navigation task, prior to the robot’s decision to stop and further move its arm to fetch an object, a tradeoff may also be desirable by distinguishing the value of the object to see if the location is worth a visit versus taking another route with possibly more valuable objects along the way with shorter distance to the goal. In this scenario, the side effects correspond to the amount and type of the objects gathered in a location and perhaps the energy consumed within the robot as a result of the action. Putting these into a whole picture, there are two levels of complexity that go beyond what a simple action set can express in a decision

process: i) realizing an action may require more than one parameter to be controlled simultaneously and ii) the localized impact from the agent performing an action is propagated to the environment step by step through the sequence of actions manifesting as a dynamic process.

Recall that in standard RL methods, all features are used to engineer the state space including those that may be directly linked to the action dynamics; as such, actions are characterized by a discrete or continuous set of decision choices that collectively serve as a mechanism of state transitions without the flexibility to express the complexity inherent in the action as it is performed. Such formulation poses a significant limitation on the expressiveness of the learning algorithm. To increase the flexibility of action specification, we consider instead using a set of parameters that identify the way they influence the external environment as a dynamic process, similar to the state abstraction with features that represent the key aspects of the environment. The actions defined through parameters are conveniently called *parametric actions*.

Further, with parameterization of actions, the assignment of features to express actions can often yield a reduction of the feature space necessary to characterize the state, which would otherwise grow exponentially as the number features are increased linearly. This is often a bottleneck in standard learning algorithms, commonly referred to as the *curse of dimensionality*. We shall use the Grid computing scenario from the earlier chapter to illustrate the challenge faced by most of the standard learning algorithms whose representation is inherently limiting in its flexibility to achieve an effective dimensionality reduction.

Back to the scenario given in the chapter beginning where a stream of user tasks are pending to be assigned to appropriate VMs running on computing nodes from federated Grid sites. The challenge arises when considering the optimization objective such as dispatching tasks to appropriate Grid nodes in a desired sequence such that the response time for tasks are minimized or that the overall resource utilization is maximized. Now, in the framework of

reinforcement learning with MDPs, task assignment to a particular machine can be modeled as one single action; therefore, a large set of target machines equate to a large set of actions. Nonetheless, these seemingly distinct actions share commonality if viewed in terms of a corresponding set of machine attributes such as CPU speed, memory, swap space, etc. In particular, the task-assignment actions with compatible machines in terms of these attributes (e.g. resource capacity being more than the task demand) will generally be distinguishable from otherwise incompatible cases (e.g. insufficient resource capacity, wrong platform, etc). Note also that the machine profile is in general non-static (e.g. through changing real-time resource capacity) and thus varies over time according to multiple threads of complex factors such as system configurations, the site's local policies, data movements, incoming workload, and available bandwidth, among others. Thus, to learn a task assignment policy with the conventional RL approach, each attribute of the machine would be required to be engineered into the state space. Suppose that each machine is represented by n attributes and a total of m machines are matching candidates for a given job request, then there will be as many as $n \times m$ features to be considered for the state abstraction in addition to possibly other features including those for the tasks and the overall statistics of the resource pool. Under this representation, the state space dimension can potentially grow very quickly in addition to a large set of task-assignment actions when m grows large. Similarly, in route planning, robot navigation, assembly line scheduling, and dynamic channel allocations in wireless communication systems, etc all share a common trait – a large set of complex actions at each decision point. These actions often have a direct influence on only a subset of the state space through the features that represent their dynamics. In the Grid scheduling scenario, the set of features that represent the time-varying property for candidate machines, are often in direct relation with the task assignment actions. These "action-bound" features, from the perspective of standard RL approach, are incorporated as a part of the state space in order to learn a scheduling policy that distinguishes individual machines. However, a linear expansion of

available compute resources would imply an exponential growth in state space. As such, it would be very difficult to realistically integrate the learning framework to a workload management system which, at any given time, may have access to hundreds or more VMs let alone learning an optimal task-assignment policy within a short time.

Parametric actions can be best modeled by features that not only characterize the action dynamics but, more importantly, allow for predicting its final outcome. Specifically, let $X_A : \{x_1, x_2, ...x_n\}$ denote the set of n features that best describe the action set $A : \{a_i \mid 1 \leq i \leq m\}$ of m actions. Under this model, each action a_i is parameterized in the form of $a_i(\mathbf{x})$, where $\mathbf{x} = \langle x_1, ..., x_n \rangle$ is an action feature vector in $\mathbb{R}^n$ and $a_i(\cdot)$ is a function that maps $\mathbf{x}$ to a desired feature space H_A (e.g. Hilbert space). In this manner, the similarity of actions can be defined geometrically, for instance, through an appropriate Euclidean distance metric or an inner product evaluated through kernels [1, 22].

Kernel functions, a branch of study from functional analysis [104], have many desirable properties that can be used in the framework of parametric action models as we shall see in the next chapter. Generally speaking, $a(\mathbf{x})$ represents a function of the feature map from the input space $\mathbf{x} \in X$ to a feature space H_A. This means that a kernel function $k(\mathbf{x}, \mathbf{x}')$ can be defined such that the similarity of two parameterized actions in terms of an inner product (i.e. $\langle a(\mathbf{x}), a(\mathbf{x}') \rangle$) can be evaluated in H_A via $k(\mathbf{x}, \mathbf{x}')$. Depending on the action dynamics in a problem domain, one could then design a valid kernel (e.g. squared-exponential kernel, etc) that induces an appropriate inner-product definition to measure the similarity of two actions in the feature space. However, in this chapter, we shall first assume that $a(\cdot)$ maps $\mathbf{x}$ to itself, namely, the same input feature vector $\langle x_1, x_2, ...x_n \rangle$ in order to focus on our first application of the parametric action model – a cluster-based action abstraction that aggregates similar actions. Later on in Chapter 5, it will become clear that any positive definite kernels can be used

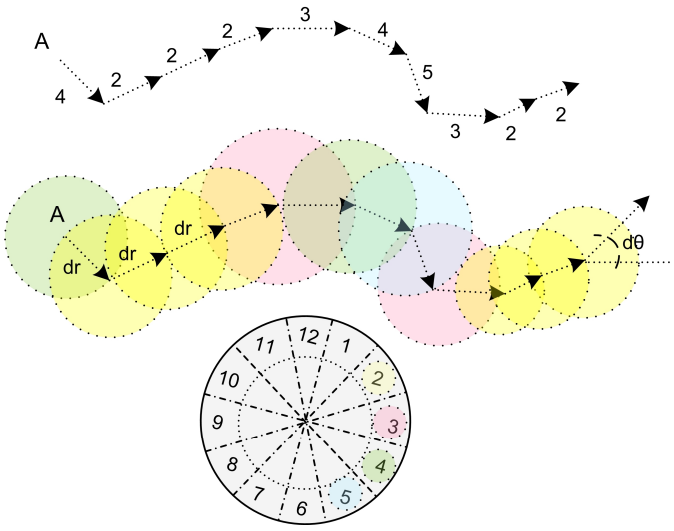


Figure 4.5 Parameterizing actions. The 12 primitive actions shown at the bottom circle are parameterized by a radius and an angle increment relative to the current position.

to achieve such a mapping that induces a valid inner product space (i.e. $k(\mathbf{x},\mathbf{x}') = \langle a(\mathbf{x}), a(\mathbf{x}') \rangle$). On a relevant note, Euclidean distance metric, for example, is strictly not an ideal measure in this context because the Gram matrix formed by a Euclidean distances as entries (or Minkowski distance measures in general) is not positive definite. Also, in Chapter 6, we shall see a generalization of the action similarity measure into a context-dependent similarity measure that takes into account the combinatorial effect from both the parametric action and its associated state feature vector (i.e. the decision context where the action was chosen).

The use of parametric actions is perhaps best illustrated with examples. In a 2D navigation domain where an agent is trying to find the best path towards the goal (e.g. shortest path), instead of specifying a concrete action set that moves the agent along a fixed direction, parameters are used instead. Figure 4.5 illustrates 12 primitive actions, each of which moves the agent in a particular direction expressed by the radius and angle increment (i.e. $(\Delta r, \Delta\theta)$) relative to the current position. In particular, Δr is bounded within the inner and outer circles while $\Delta\theta$ is bounded within a pie-shaped partition depending on the action chosen. These

parameters effectively specify a feasible range of motion per move. This type of action parameterization captures the stochastic property of actions due to the uncertainty in the environment including a possible misalignment in the agent's actuators. Note that the term primitive action is used to describe a concrete, executable action, which is to be distinguished from an abstract action in the later discussion.

Other possible action parameters under a navigation domain include, for instance, features that capture the "side effect" of visiting each location such as the reduction of particular objects collected by the agent upon visit. In this case, the state is not only characterized by the current position of the agent but also by the property of objects available in that position. An example for this is a planetary exploration domain where the goal is not only reaching a particular destination but also gathering maximal natural resource of interest by following a relatively shorter path to minimize energy consumption. A simple simulation of the planetary exploration domain is given in the last section of this chapter.

4.4 Forming Our First Abstract Action Model

We will explore in this section how parametric actions can be applied to the framework of standard reinforcement learning algorithms using Q-Learning as an example.

The first step of defining a parametric-action model is to identify the features that distinguish one action from another and that also contribute to (or allow to predict) the final outcome. The principle of selecting action parameters is similar to the feature selection for the state space representation but yet with a subtle distinction. Action parameters are meant to represent a "local variation" in response to the action taken in the environment while state features altogether remain as a representation for the environment perceivable by the agent. For instance, the resource variation in a single VM node is local with respect to the entire resource pool and similarly, a position increment in the navigation domain is local to the current position compared to the scale of all the reachable state space. This is similar in a way to the

primitive action set in MDPs where each action also effectively specifies a local shift in the state space in that taking an action triggers a state transition from the current to the successor state, which is indeed localized compared to the entire chain of the state sequence.

With action parameters defined, it is then possible to learn an abstraction over actions that share similar properties. This opens up the possibility of reducing action choices at each decision point and also of modeling the impact to the environment caused by applying an action (e.g. removal of certain objects upon visit of a location) as a dynamic process. To start with, we will first define the notion of *action influence* that represents the functional relation between the action parameter and the action's final outcome in terms of two different measures: i) immediate return – the value returned from the reward function $R(s,a(\mathbf{x}))$ and ii) accumulated reward – the utility obtained from the estimated action-value function $Q(s,a(\mathbf{x}))$, where $s \in S$ and $a \in A$ as mentioned earlier in Section 4.1 except that a is parameterized by $\mathbf{x}$.

A high degree of similarity for a group of actions implies that taking these actions leads to approximately the same reinforcement feedback (bounded by a small variance σ) and these actions are said to share similar action influence over the environment. We will first introduce the following assumptions for the convenience of the initial discussion: i) similar actions are identified through similar immediate returns ii) the functional relation from the action parameter to the reward is given as a heuristic. Note that using the immediate return as a measure of action similarity is only feasible when the action parameters are invariant under state transitions or at least only vary within a small range. In other words, the action parameter under this naive assumption effectively models a global view of parametric actions with respect to all states. The general case, however, would require using the accumulated reward (or utility) as the action similarity measure by distinguishing the state in which parametric actions are applied as well as the state transition dynamics. Only then can we predict the utility of applying a particular parametric action in an arbitrary state, thereby obtaining a complete and well-defined decision process. Nonetheless, this requires a more advanced treatment through a sequential model

selection by adjusting the action similarity measure (e.g. kernel) in response to the progressive update on Q-value estimates. Further, the desired clusters that associate similar actions need to account for the states in which they were applied in terms of the functional relation between the following two constructs: i) a joint representation for the state and the parametric action of choice; i.e. a *generalized state-action pair* and ii) the value estimate for the generalized state-action pair. In other words, the cluster reflects not only the similarity of state-action pairs but also their functional responses. Here, to simplify our initial use case for parametric actions, we shall first introduce a much simpler yet useful cluster-based action model by assuming that appropriate action parameters and their corresponding similarity measure are provided heuristically as mentioned earlier, and that the action similarity measure is static (i.e. non time-varying). We will remove these assumptions after the necessary background is introduced in the later chapters.

4.4.1 Action Abstractions with Clustering Mechanism

Now we will examine the action clusters formulated by grouping actions that share common action influence in terms of immediate reward. With action clusters, the policy learning process, by implication, involves two levels of resolutions: i) a policy in terms of action clusters (i.e. in the unit of abstraction) and ii) a policy for resolving an individual action from a given cluster. Specifically, let $A_c : \{A^{(i)} \mid 1 \leq i \leq k\}$ denote a set of k clusters, each of which contains similar actions induced by a clustering algorithm (e.g. k-means) over the action parametric space.

For the action-value learning algorithm such as Q-learning, the first stage is to evaluate the high-level policy defined over the action cluster $A^{(i)} \in A_c$. This can be represented by the following equation as a generalization over (4.11):

$$Q(s, A^{(i)}) \leftarrow Q(s, A^{(i)}) + \alpha\left[r + \gamma \max_{A^{(j)}} Q(s', A^{(j)}) - Q(s, A^{(i)})\right], \tag{4.12}$$

where $\pi(s)$ returns the best action cluster $A^{(i)}$ that maximizes $Q(s, A^{(i)})$ under the current estimate of the policy. The second stage is to resolve the best primitive action $a_j^{(i)} \in A$ from $A^{(i)}$. For the convenience of exposition, the process of resolving an action from a given cluster is defined as *declusterization*. The strategy for declusterization can range from random selection to domain-specific heuristic and automated policy learning. In Chapter 7, after sufficient background coverage in kernel methods, we will explore a kernel-based pattern-matching strategy. For the moment, the *cluster centroid* is used, assuming that the centroid can effectively serve as a reasonable generalization over all other action members from the same group.

Grouping actions with similar action influence in their parametric space can result in a significant reduction of the state space required to represent the domain. Formally speaking, suppose that $X_S : \{x_1,...,x_p,x_{p+1},...,x_{p+q}\}$ denotes the entire feature set in the state space (of dimension $p+q$) and suppose further that the actions $a_i(\mathbf{x}_a) \in A$ are parameterized by the features in the set $X_A : \{x_1, x_2,...x_p\}$, where $X_A \subset X_S$ and $\mathbf{x}_a$ is the action vector with vector components assuming values of those features in X_A such that any $\mathbf{x}_a$ has dimension *p*. Now, assume that we have clustered the action set into *k* distinct groups denoted as $A_c : \{A^{(1)}, A^{(2)},..., A^{(k)}\}$. With the formulation above, clustering actions can effectively reduce the state space to, $X_S' : \{A^{(i)}, x_{p+1},..., x_{p+q} \mid A^{(i)} \in A_c\}$, which can be a significant reduction provided that $k \ll |X_a|$. This is usually the case as long as action vectors induced by X_a are not purely random (and thus can be properly clustered).

Note that the notation $A_c : \{A^{(i)} \mid i = 1,...,k\}$ as an abstract action set is chosen to distinguish from the primitive action set $A : \{a_j(\mathbf{x}_a) \mid j = 1,...,n\}$. In addition, $|A_c|$ is usually taken to be smaller than $|A|$ (i.e. $k \leq n$) such that each cluster references more than one primitive

action member. As an example, suppose there are n candidate machines, $\{m_1, m_2, ...m_n\}$, on which a particular task J_i can be executed at time t. Meanwhile, suppose that, the machine m_1 is characterized by the following three different attributes: a CPU speed of 2GHz, an available memory of 1G bytes and a remaining disk space of 200G bytes. Given the aforementioned resource profile, the action of assigning tasks to m_1 can be represented by $a_1(\mathbf{x}_a) = \langle 2G,\ 1G,\ 200G \rangle$. Similarly, assume that the action of assigning tasks to m_2 with a slightly different resource capacity is represented by $a_2(\mathbf{x}_a) = \langle 2.23G,\ 0.92G,\ 198G \rangle$ and the assignment to m_3 as $a_3(\mathbf{x}_a) = \langle 1.0G,\ 500M,\ 200M \rangle$, which indicates a significantly different resource profile than m_1 and m_2. Since the machine profile of m_1 is relatively closer to m_2 (as opposed to the difference between m_1 and m_3 in their resource capacity), it is very likely that the time it takes to finish task J_i on m_1 (i.e. execution time on m_1) will be very close to the time to finish the same task on m_2, assuming that the task is compatible with the candidate machines. Hence, if the reward is defined as a function of execution time, then the final outcome from taking action a_1 and a_2 will be approximately identical. This implies that they can be represented by a cluster that contains possibly other machines with similar properties. With an appropriate similarity measure defined over the action feature vector, one can predict a reasonable degree of similarity over a set of actions in regards to their future outcomes. Consequently, if a decision process contains a large set of actions, then those with similar action influence over the state space (determined by action parameters) can then be grouped together in order to reduce the complexity in decision making.

However, there is a caveat in using cluster-based abstraction over actions. It is possible that the declusterization process resolves to an empty set since the particular action group of interest (i.e. the action cluster that leads to maximum reward) might have become exhausted during incremental adjustments of cluster members whose action dynamics, namely, parameter values are expected to change over time. We shall see in Chapter 7, a more rigorous

way to resolve this issue. Here, we shall resort to an approximate solution by continually choosing the second-best cluster (when the previously best action cluster is currently empty) until a particular parametric action can be resolved.

4.4.2 Incremental Cluster Adjustments

The fact that the action dynamics modeled by the vector $\mathbf{x}_a$ (i.e. the parameter vector associated with action *a*) may vary over time, from the clustering standpoint, can lead to variations in the cluster structure. Using the earlier example in the task assignment domain, the resource capacity such as the available memory at m_1 and m_2 may be close to each other at time *t* but the memory profile will very likely fluctuate as they both take on incoming user tasks, leading to a possibly larger gap in the future memory profile. Subsequently, it leads to a decreasing degree of similarity in the task-assignment action over the machine m_1 and m_2. Perturbations in the similarity degree of the action influence over the state space indicate that action clusters formulated in the past may not be strictly applicable for the future scenario without some alteration of the existing clustering structure. This can be addressed, for instance, by incremental cluster adjustments using locally weighted k-Nearest Neighbor (kNN) [100]. At the very beginning, an initial clustering structure over actions is established by a clustering algorithm appropriate for the problem domain. K-means++ [52] is among the ideal choices due to its potentially better clustering result through a careful seeding over cluster centroids prior to the standard K-means procedure. That is, the resulting cluster structure, with higher probability, minimizes the distortion of cluster members (e.g. minimum total sum-of-square distance with respect to the cluster centroid). With an initial clustering structure determined, kNN then performs subsequent cluster re-assignments in response to the variation in action feature values. Specifically, following an update of the action feature values from $a_i(\mathbf{x}_t)$ to $a_i(\mathbf{x}_{t+1})$, the *k* nearest neighbors are selected from the parameterized action set *A* to determine the majority vote for relocating $a_i(\mathbf{x}_{t+1})$ to an appropriate cluster. In addition, depending on selected action

features and their correlations, it may be desirable to design a kernel function that transforms the distance between action vectors in the original input space to an appropriate similarity measure that assign appropriate weighting to each feature dimension. We will have more to say regarding the technique to adjust the feature weighting in Section 4.4.3.

Further, as clusters are progressively updated, action feature values may fluctuate back and forth, in the extreme case leading to empty clusters. It is thus necessary to partially retain the old clustering structure to allow the varying actions to possibly regain their old cluster affiliations. To achieve this, selected parametric actions with their corresponding stale parameter values, defined as *pseudo data points*, are retained to serve as references for the kNN procedure to possibly reassign updated actions to the old clusters. Specifically, when an action $a(\mathbf{x}_t)$ is relocated from, say, cluster $A^{(i)}$ to $A^{(j)}$, then the older position in $A^{(i)}$ is retained as a pseudo data point. Conversely, later on when a new action member joins cluster $A^{(i)}$, then the closest pseudo data point is removed. If there exist no pseudo data in $A^{(i)}$ upon the merging of an action, then no data removal is required. In addition, a new global readjustment is performed (e.g. applying a new cycle of Kmeans++) when new action clusters no longer align with the new data distribution (i.e. when the amount of pseudo data points grows larger in proportion to the currently active parametric-action members). Lastly, pseudo data points are removed after new action clusters are formulated. In Chapter 7, we shall see a cleaner solution using a non-parametric, data-driven learning method to deal with the progressive morphing of cluster structures. Figure 4.6 illustrates the resulting combination of Q-learning and parametric action clustering as CAQ-learning algorithm (i.e. Clustered-Action Q-learning).

Note that in order to formulate appropriate action clusters, a cluster assumption is required as an input to CAQ-learning. This assumption over action similarity is denoted by the function h_a in Figure 4.6 For instance, if X_A contains p features, then h_a is a function that maps a *p-vector* to a real number; i.e. $h_a : \mathbb{R}^p \rightarrow \mathbb{R}$. In the subsequent chapters, we will

CAQ_Learning (X_S, X_A, h_a):

// X_S : state feature set

// X_A : action feature set and $X_A \subseteq X_S$; each $a \in A$ is parameterized using X_A so that

// actions are explicitly represented in vector form: $a(\mathbf{x}_a) \in A$, where each

// coordinate $(\mathbf{x}_a)_i$ takes on the value of the *ith* feature; i.e. $(X_A)_i$.

// h_a : heuristic over action similarity that maps $a(\mathbf{x}_a)$ to a real number (i.e. cluster

// assumption): e.g. let $a(\mathbf{x}_a) = \mathbf{x}_a$ and $h_a \triangleq \sum_i (\mathbf{x}_a)_i$.

1. **Initialize** action clusters $A_c : \{A^{(1)}, A^{(2)}, ..., A^{(k)}\}$ using a selected instance-based clustering algorithm (e.g. k-means++).
2. **Initialize** $Q(s, A^{(i)})$, $A^{(i)} \in A_c$ arbitrarily.
3. **Repeat** for each episode:

   3.1 **Choose** $A^{(i)}$ from s using policy derived from the Q-value over action clusters (e.g. using ε-greedy exploratory strategy)

   3.2 **Decluster** $A^{(i)}$ to resolve a primitive action $a_j^{(i)} \in A^{(i)}$ using a selected action-resolution policy. E.g. using centroid action:

   $$a_j^{(i)} \leftarrow getCentroid(A^{(i)} \in A_c)$$

   3.3 **Take action** a (i.e. $a(\mathbf{x})$) and observe r, s', $a(\mathbf{x}')$, then use the immediate reward r to update the Q-value for $A^{(i)}$ using the following update rule (Eq. (4.12)):

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

   If the dynamic of $a(\cdot)$ changes; i.e. $a(\mathbf{x}_t) \neq a(\mathbf{x}_{t+1})$

   Then:

   Apply incremental cluster adjustments with ***weighted KNN*** *and* ***pseudo data points***

   3.4 **Update** to the new state: $s \leftarrow s'$

Until s is terminal

Figure 4.6 CAQ-Learning. Policy is derived over the set of abstract actions based on clustering similar actions that lead to approximately equal action influence.

progressively establish a more full-fledged learning method that automatically identifies the correlation between state/action features and their corresponding function responses (e.g. utility) through the policy learning experience. Nonetheless, the functional response is restricted to the instantaneous reward for the moment. An example of designing the action cluster assumption will be given shortly in the following sections.

4.4.3 Feature Selection in Cluster-based Action Abstraction

One caveat of cluster-based action abstraction determined solely by their associated action features is that the accuracy of the clustering results is highly contingent on how, in actuality, these features are related to an action's final outcome (even before considering their variations with respect to states). Choosing irrelevant features to model actions will certainly lead to an undesirable definition of action clusters. A more involved clustering approach is to gradually fine tune the weighting of the chosen features with respect to the final returns of actions. The weighting indicates how relevant a particular feature is to the final return of an action carried out in the state space. After the weight adjustment, the similarity measure, if converged, will be tuned to the form with a desired weighting of features, leading to a cluster structure that better reflects the action returns. To achieve this, one possible solution is to express the similarity measure in the following form:

$$d(\mathbf{x},\mathbf{y}) = \sum_{i=1}^{p} w_i \left| x_i - y_i \right|, \tag{4.13}$$

where w_i denotes the weighting of the *ith* action feature and $\mathbf{x},\mathbf{y}$ are both objects represented as feature vectors of p dimensions. In the task assignment domain, for a CPU-bound task, the CPU speed and available CPU time will both influence the task execution time more significantly than the disk space. Conversely, the disk space and perhaps memory are more influential for an IO-bound task. Thus, CPU-related action parameters should generally have higher weightings in (4.13) than the other features for CPU-bound tasks. This type of clustering approach is called *adaptive clustering* [54]. Research on adaptive clustering algorithms using external feedback (e.g. the immediate return of an action) to adjust feature weights has shown promising results in determining reasonable clusters when compared to those from other classifiers such as NCC (Nearest Centroid Classifier) [98]. However, the computational expense of running an adaptive clustering algorithm in the parametric action space may outweigh the benefit of compressing action and state space since the underlying clustering

algorithm (e.g. k-means) needs to be iteratively executed numerous times in the search space before reaching optimal feature weights. When the parametric space of actions fluctuates to a great extent, the entire clustering procedure may have to rerun again in order to identify the new correlation between the feature weighting and the true action return.

At this point, a cleaner solution may not be readily available as it would take a few steps further to automate the feature selection procedure and in the meantime, integrate it into the reinforcement learning framework. That is, the integration should enable policy learning and action model selection to simultaneously take place and mutually leverage the strength of one another. In the later chapters, we shall introduce a new concept-driven learning architecture (CDLA) that accomplishes the aforementioned aspects in parallel through three types of interleaving processes: policy learning, clustering and regression. In particular, CDLA employs Gaussian Process Regression (GPR) [14] as a utility predictor and a generalization mechanism that guides the control learner in policy learning while using the *spectral clustering* [24] approach to formulate abstract actions as high-level control decisions. Essentially, the Gaussian process drives the clustering procedure to aggregate similar parametric actions by taking into account their correlations with states based on an adaptable action-state correlation hypothesis – kernel functions used in both the Gaussian process and the clustering algorithm. In this new learning framework, as we shall see, the optimization of the weighing of action parameters and additionally state features can be achieved through maximizing the marginal likelihood of the data.

4.5 A Navigation Domain

To demonstrate the potential usage of CAQ-learning, a navigation domain is designed. Here, a rover is sent to a planet to collect rare chemical substances hardly available on the Earth. Yet, it is costly to explore all arbitrary regions on the planetary surface since these elements are only richer at limited geographical locations. These locations are sporadic

throughout the entire planet. To be time and energy efficient, it is imperative to develop an ideal route planning strategy in order to gather the largest amount of the sought-after chemical elements with minimal cost in terms of time and the rover's internal energy consumption.

This domain can be converted to a grid world by partitioning the planetary surface into several discrete grid cells. To relax geological constraints (e.g. terrain-specific obstacles), the rover is assumed to have the freedom to move from one grid cell arbitrarily to another cell, resulting in a vast choice of exploratory strategies. Since the primary goal here is to collect chemical resource, the reward function can be defined as:

$$R(s,a) \simeq R(a(\mathbf{x})) = R(a(\langle x_1, x_2, ... x_n \rangle)) \tag{4.14}$$

where $\mathbf{x}$, or $\langle x_1, x_2, ... x_n \rangle$ represents the quantities of the n chemical element of interest. Notice that $R(s,a)$ is approximately equal to $R(a(\mathbf{x}))$ but not identical from the earlier assumption that a global action model is shared by all the states but with subtle differences that are not to be explicitly addressed yet in this chapter. In the general case, however, action parameters are state-dependent but for the moment this factor is assumed to be insignificant in order to focus on the effect of action clustering as an abstraction. Additionally, with the assumption that $a(\cdot)$ is a function that simply maps the input to itself, (4.14) can be further reduced to $R(\langle x_1, x_2, ... x_n \rangle)$. As a reasonable assumption for this domain where the reward increases in proportion to the amount of chemical elements gathered, $R(\cdot)$ can be expressed as:

$$R(\mathbf{x}) = R(\langle x_1, x_2, ... x_n \rangle) = \sum_{i=1}^{n} w_i x_i, \tag{4.15}$$

where w_i indicates a weight for each element. Assuming further that all elements are equally important, giving $\{\forall i, w_i = c \mid c \in \mathbb{R}\}$, the reward is effectively determined by the sum of all chemical elements gathered (scaled by a constant). In this manner, clustering similar actions amounts to clustering target cells that share a similar chemical distribution (with respect to the net value of chemical resource) and thus lead to approximately the same immediate reward. In

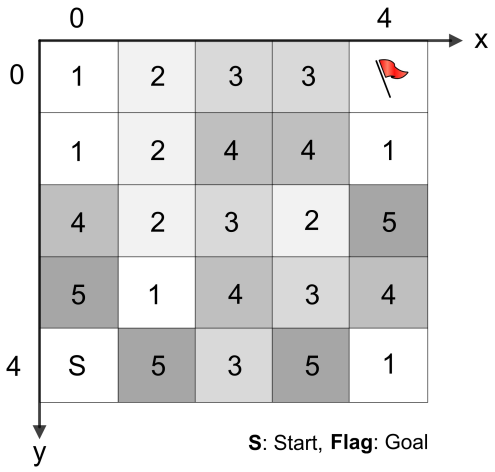


Figure 4.7 A Gridworld representation for the planetary navigation domain. This simulated planetary surface is covered with REEs in different grid cells. Those cells marked with the same number contain approximately the same amount of REEs in terms of their net total.

particular, Manhattan distance can be used here as the similarity measure since the difference between any two actions are only dependent on the sum of elements, which by (4.15), distinguishes the action outcomes. For the simulation, a 5-by-5 grid world is generated with 5 distinct patterns of the distribution over 3 different rare earth elements (REEs) *Eu*, Yb, and *Lu* whose quantities are respectively used as action features. Each chemical pattern is generated with a predefined distribution of chemical elements modulated by Gaussian noise. The selection of the grid cells following the same distribution pattern is performed via a uniform sampling that favors each pattern equally over all grid cells and thus, for a 5-by-5 grid world, approximately 4 to 5 cells follow the same pattern of chemical distribution (except that the start and goal cells are assumed to have no chemical elements). In this manner, clustering over the distribution of chemical elements with respect to their quantities correspondingly results in 5 distinct groups of action clusters (i.e. $A_c : \left\{A^{(1)},...,A^{(5)}\right\}$). The idea is illustrated via Figure 4.7 where cells of the same number indicate similar pattern of chemical bearings. Actions that lead to these areas are expected to receive similar amount of rewards. A summary of the specification for the experiment is presented in Table 4.1.

Table 4.1 Specification for the planetary navigation domain.

WORLD	*Gridworld dimension*: 5×5 *S: start state* (see Figure 4.7) *Flag: goal state* (see Figure 4.7) *Number of distinct patterns of REE distribution*: 5 *Number of grid cells following each pattern*: 4 ~ 5 ***Maximum accumulated reward*: approximately 985.0** *Initial REE quantities averaged over the grid cells with richest chemical resource*: {*Eu*: 32.24±1.14, *Yb*: 48.80±0.57, *Lu*: 16.30±0.88} *Initial REE quantities averaged over the grid cells with scarcest chemical resource*: {*Eu*: 0.50±0.57, *Yb*: 1.04±0.71, *Lu*: 0.01±0.01(0.007)}
REWARD	*Positive reward from collecting REEs*: Sum of element quantities. *Negative reward due to energy consumption per step while rover has sufficient energy*: -1.0 × Manhattan distance (from start cell to target cell) *Negative reward due to energy consumption per step when using backup energy*: -100.0 × Manhattan distance
BASELINE RL ALGORITHM	Q-learning with $\varepsilon-$ *greedy* *Learning Rate*: 0.09 *Future Discount*: 0.92 *Initial* ε : 0.8
INCREMENTAL CLUSTERING	Alternating learning cycles between *K-means++* and incremental adjustments via *weighted kNN* and *pseudo data points* (see Section 4.4.2)
ACTION SIMILARITY MEASURE	Manhattan distance over feature $\langle \theta_{Eu}, \theta_{Yb}, \theta_{Lu} \rangle$; i.e. $\sum_{i \in \{Eu,Yb,Lu\}} \theta_i$

One caveat is that, as the rover travels through grid cells, energy is consumed and should be counted negatively towards the overall payoff. The energy factor indeed makes the action-model explicitly dependent on states as in the general case since the further distance

Table 4.2 State features and action parameters for the navigation domain.

STATE FEATURES	$X_S : \{x, y, \theta_{Eu}, \theta_{Yb}, \theta_{Lu}, \textit{hasEnergy}\}$, where $\langle x, y \rangle$: Coordinates of grid cells $\langle \theta_{Eu}, \theta_{Yb}, \theta_{Lu} \rangle$: Element quantity (action parameters; see below) *hasEnergy:* A predicate indicating if rover has energy
ACTION FEATURES	$X_A : \{\theta_{Eu}, \theta_{Yb}, \theta_{Lu}\}$, where θ_{Eu}, θ_{Yb}, and θ_{Lu} denote the quantity for *Eu*, *Yu* and *Lu* in each grid cell, respectively

away from the goal implies more future energy consumption. Nevertheless, with the assumption that the priority of gathering chemical elements far outweighs the need of careful energy conservation, the clustering result based on chemical features will still yield a reasonable grouping of actions. This representational approximation is compensated for by imposing a strong penalty per step if the energy runs out before the rover reaches the goal. If desired, the trade-off with respect to traveling distance can be incorporated into the secondary policy (i.e. declusterization) in which a primitive action is chosen in favor of the closest grid cells out of those with similar chemical distributions. Yet, in a general domain, it may not be straightforward to obtain a good heuristic for the declusterization process in this manner. Accurate action modeling in this regard will be studied in upcoming chapters. As discussed earlier, clustering similar actions leads to a compression of the state space. As shown from Table 4.2, the state feature space is characterized by cell coordinates $(x, y) \in X \times Y$, quantity of each chemical element $\boldsymbol{\theta} \in \theta_{Eu} \times \theta_{Yb} \times \theta_{Lu}$, and a predicate indicating if the rover still has sufficient energy left to travel. For ease of exposition, let A^+ denote the action parameter space formed by the Cartesian product of features in X_A to distinguish it from the primitive action set A; similarly, the state space S is represented by the Cartesian product of features in the set X_S (a superset of X_A). Then, if A^+ consisting of all possible combinations of features $\{\theta_{Eu}, \theta_{Yb}, \theta_{Lu}\}$

can be compressed to $A_c : \left\{ A^{(i)} \mid i = 1,...,5 \right\}$ with 5 distinct action clusters, then S is by implication reduced from $S : X \times Y \times A^+ \times \{True, False\}$ down to the level of $S' : X \times Y \times A_c \times \{True, False\}$, where $X \times Y$ refers to the set of cell coordinates.

Depending on the problem domain, it is possible to further compress the state space on some occasions where the action parameters are correlated to other subsets of state features. In this experiment, notice that the property that differentiates each grid cell is the unique pattern of chemical distribution characterized by the cluster label $A^{(i)} \in A_c$, each of which corresponds to a set of locations (or jointly a region) sharing a similar chemical constitution. Formally speaking, there exists a functional mapping F such that $F : X_S \rightarrow G$ where $G \subseteq X \times Y$. Consequently, S' in this case can be further reduced to $S'' : G \times A^+ \times \{True, False\}$. This implies that upon any Q-value update, a set of grid cells sharing a similar pattern of chemical distribution can be updated altogether, further sharpening the learning curve.

4.5.1 Experimental Results

Figure 4.8 shows the learning curve, in terms of accumulated reward per episode, obtained from CAQ-learning with 5 action clusters, equal to the number of actual patterns of the underlying chemical distribution. Each symmetric error bar along the curve indicates one standard deviation above and below the average in accumulated reward. As the curve indicates, CAQ-learning with an appropriate exploratory strategy converges to a nearly optimal policy (with accumulated reward approximately 985). An optimal policy would lead the rover to gather as much chemical resource as possible, following a shortest path to the exit, before its energy runs out. However, in order to approximate the optimal policy better, it is required that the difference in cell locality be considered in the action clustering procedure, yielding more accurate estimates of action influence, since traveling to each grid cell requires different amounts of energy.

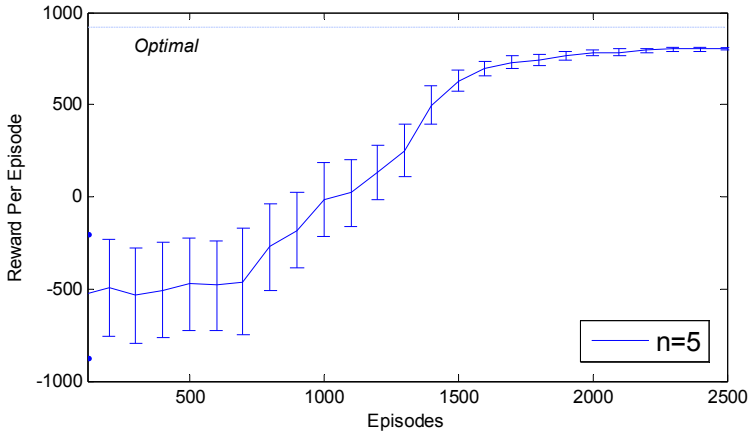


Figure 4.8 Learning with action clusters. Number of clusters: n = 5.

Figure 4.9 illustrates the learning curves for 2, 5, and 8 action clusters with a focus on later learning episodes and their convergence properties. As the result indicates, learning with a number of action clusters smaller or larger than the number of actual underlying patterns (5 patterns as mentioned earlier) results in a lower accumulated reward. Smaller cluster numbers lead to a less accurate estimate of action influence, leading to “ballpark decisions” along the way whereas larger cluster numbers require more exploration in order for the learner to identity better decisions before settling on a greedy policy. An extreme case with a large cluster set is learning with primitive actions themselves. This is presented by Figure 4.10, which compares the learner using primitive actions with the one using action clusters under the same exploratory condition (i.e. the same decay rate of the exploration probability). As the experiment suggests (Figure 4.10), learning with primitive actions levels off to a much lower accumulated reward than with clustered actions. This is due to the fact that using primitive actions for the control policy requires longer training period before the policy converges.

That learning without associating similar actions is much more computationally expensive for this domain can be illustrated with two closely-related observations: i) the rover has here approximately as many choices of actions as the size of the environment, and ii) no

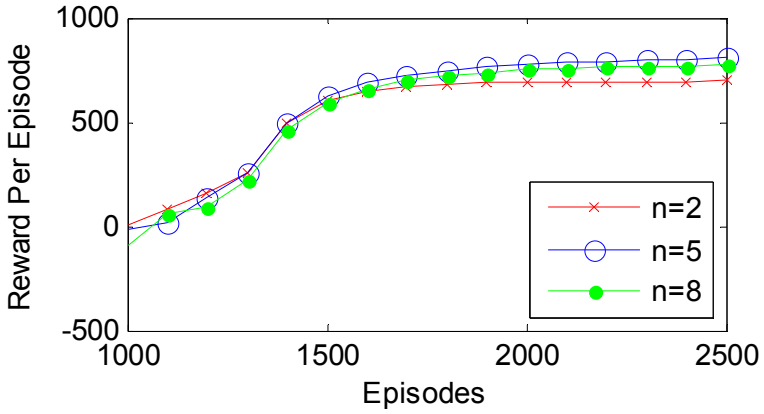


Figure 4.9 Comparison with different sizes of action clusters: n = 2, 5, 8.

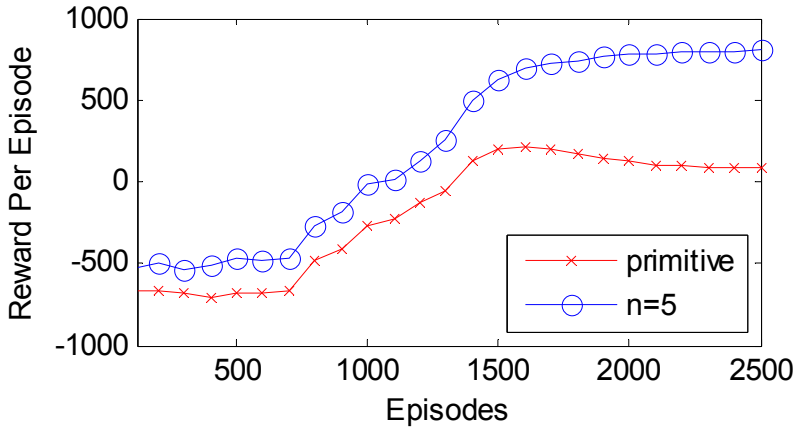


Figure 4.10 Learning with clustered actions (n=5) versus primitive actions.

action constraints are present as the rover freely selects any of the grid cells to travel to, yielding a set of possible state transitions that grow quadratically in the size of the state space; i.e. in the order of $O(|S|^2)$.

CHAPTER 5

SIMILARITY, KERNEL AND FUNCTION APPROXIMATION

In order to further generalize the cluster based action model introduced earlier, we will investigate in this chapter a general perspective of the regression problem from which to obtain necessary building blocks toward the required representation for policy learning in a highly versatile environment. The objective is to find an adaptive regression model that can reasonably approximate the value function in the RL framework and in the meantime provides a clean representation that enables a quantitative comparison between decisions made across the state space. In particular, given the parametric-action model from Chapter 4, the regression model needs to be equipped with a generalization ability that allows for value predictions to occur over arbitrary pairings of the state and the parametric action, from which a control policy can then be derived. That is, we wish to achieve value prediction in parallel to the learning of a similarity measure over the generalized state-action pair in which the action is represented in a vector form. In this manner, the learned similarity measure can be used to compare localized policies exercised earlier and subsequently serve as a cluster assumption to build high-level abstractions.

We will progressively build the background of the generalized RL framework through the following four conceptual pieces in the later discussion: (1) a basic outline of regression analysis (2) kernels (3) Gaussian process regression, and (4) spectral clustering. In this chapter, we shall first motivate the use of kernel functions in developing an adaptive regression model and also introduce related notations. Indeed, kernel functions will serve as crucial role in connecting all the logical pieces in the generalized reinforcement learning framework. Kernel functions are depicted in terms of small circles attached to each component in the architecture

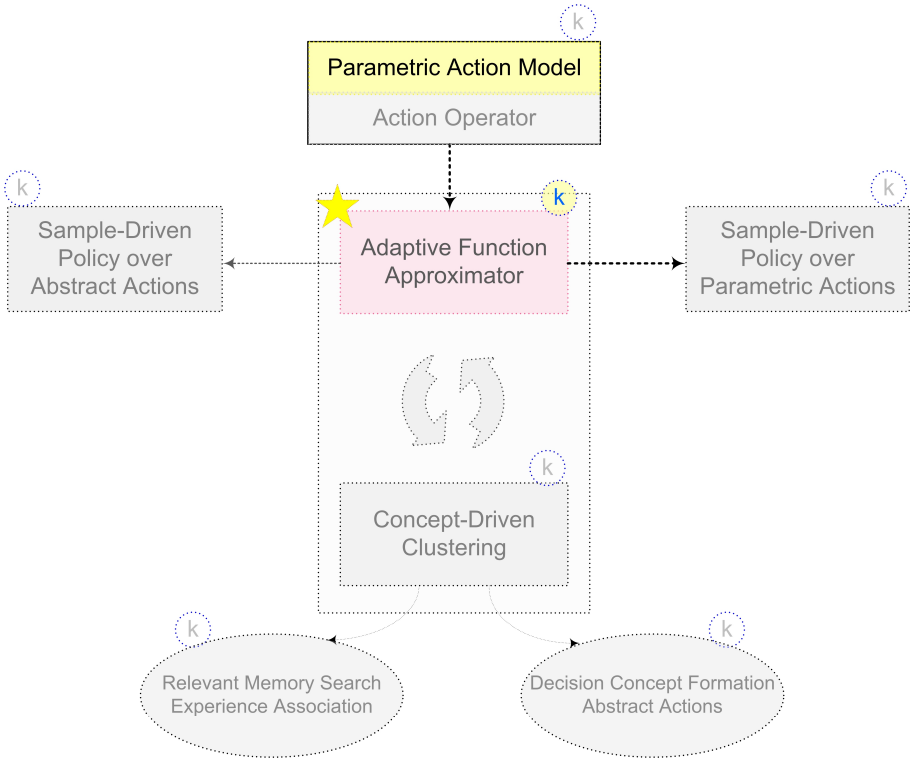


Figure 5.1 Value function approximator in a generalized RL framework. Notice that each component in the system is now accompanied by a small circle with a letter *k*. Circled-k symbols are used to indicate that the referenced components assume a representation that uses kernel functions as a building block. Simply speaking, these logical components can be or are "kernelized." This chapter will demonstrate the connection between kernel functions and function approximators.

schematic in Figure 5.1, which is inherited from the earlier chapters (see Figure 3.1 and Figure 4.1). Both Gaussian process regression and spectral clustering can be viewed as applications of kernel functions and their use in generalizing the RL framework will be covered in Chapter 6 and 7, respectively. Readers who are familiar with kernel methods in general may skip this chapter and proceed directly to Chapter 6.

5.1 Similarity Measures and Regression

In the vast machine learning domain, a common problem to solve is to discover structure in the data. Suppose we are given a training set Ω of m observations

$\{(\mathbf{x}_1, q_1), ..., (\mathbf{x}_m, q_m)\} \subseteq X \times Q$, where $\{\mathbf{x}_i\}$ denotes the set of input vectors (covariates) and $\{q_i\}$ denotes the set of scalar outputs as targets (or dependent variables). Each input vector could represent a meaningful pattern in a particular domain such as an image or a word in the document. Further, assume that each input vector has a dimension *n*, i.e. $\mathbf{x}_i \in \mathbb{R}^n$. The goal is to predict the sequence of outputs $\mathbf{q} = (q_{m+1}, q_{m+2}, ...)$ for previously unseen patterns represented by the vector set $\{\mathbf{x}_j \mid j = m+1, m+2, ...\}$. That is, we wish to obtain the conditional distribution of the target given the input data. This is only feasible if some measure is given that tells us how the new sample $(\mathbf{x}_j, q_j)$ is related to the training set. Informally, through this measure, we would expect similar inputs to produce similar outputs. To formalize this, it is necessary to define what it means to be similar. One of the most frequently-used similarity measure is the Euclidean distance (or two-norm distance) between two vectors; i.e. $\|\mathbf{x}_i - \mathbf{x}_j\|_2$. One would expect that the smaller this distance measure is, the more similar the two vectors are. This is how the majority of instance-based clustering methods express the cluster assumption: a high-quality cluster should enclose a set of input data as similar as possible in terms of the given cluster assumption, say two-norm distance.

Notice that the square of the Euclidean distance $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2$ can be re-expressed in terms of the dot product; i.e. $(\mathbf{x}_i - \mathbf{x}_j)^T (\mathbf{x}_i - \mathbf{x}_j)$. Indeed, the concept of norm can be defined in an inner product space – a vector space equipped with a well-defined inner product. However, the typical inner product definition has limited expressiveness and does not generalize to interesting yet complicated problem domains in which solutions often live in a more complex hypothesis space. For instance, in the case of the sample prediction mentioned earlier, one of the simplest ways to make a prediction would be to fit a linear function that goes across the training samples as "tight" as possible, which can be defined in terms of minimum sum-of-squared errors. In this case, the linear function is our hypothesis made with the hope that the

unseen samples will follow the same linear pattern, based on which the future predictions will be made. Such a linear function can often be expressed in the form of the dot product of the training examples:

$$Q_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

$$= \sum_{i=1}^{m} \alpha_i q_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b, \tag{5.1}$$

where in (5.1), $\mathbf{w}$ denotes a weight vector and b denotes a constant serving as a bias. In addition, $\alpha_i q_i$ as we will work out later represents a scalar in the *dual representation* of the hypothesis function $Q_{\mathbf{w}}(\mathbf{x})$ where q_i is the target for the input $\mathbf{x}_i$ in the training set Ω mentioned earlier. Thus, a prediction over a new input $\mathbf{x}$ involves a comparison with each training sample $\{\mathbf{x}_i \mid i = 1,...,m\}$ through a linear combination of their dot products: $\langle \mathbf{x}_i, \mathbf{x} \rangle$. As (5.1) suggests, the predictive value for the new data $\mathbf{x}$ is determined largely by the neighboring training samples in terms of the dot product that "weighs" heavier over the corresponding target q_i. While the intuition behind the linear model does serve as a good foundation for building more complex models, in general this often yields large prediction errors as the training sample may very well be sporadically distributed with highly irregular patterns not expressible simply by a linear combination of dot products in this form. Nevertheless, as we shall see, a similarity assumption in the form of a dot product in the vector space V is a fundamental "constituent" for a *linear smoother*, which essentially takes the form of a linear combination of the training set targets (e.g. $\{q_i\}$), each of which is weighted by a more generalized dot product – inner product. Inner product is essentially a generalization of the dot product in the vector space of $\mathbb{R}^n$ such that the inner product when considered as an operator $\langle \cdot, \cdot \rangle$ can take on operands such as complex vectors and complex functions. In addition, similar to the property of dot product in the ordinary vector space of $\mathbb{R}^n$, $\langle \cdot, \cdot \rangle$ assumes the property of being positive, symmetric and bilinear. Any positive-definite kernel (i.e. Mercer kernel), to be discussed shortly

in Section 5.3, can be shown to correspond to an inner product in the kernel-induced feature space (a Hilbert space). Also, here in this research, we will focus on the inner product over both vectors and functions in the real-valued domain. Henceforth, we shall use the term inner product to be general.

Back to the context of reinforcement learning, the subject of interest would be to predict the utility value (q_i) given a set of training samples $(\{\mathbf{x}_i\})$ in terms of a set of state-action pairs. In particular, the set of state-action pairs can be seen as a sequence of context-dependent decisions made by the agent where the context is captured by the state and the decision corresponds to the action taken. Note that we use the terminology *context-dependent decisions* (or decision context) here to lay out a possible generalization over a state-action pair in the standard RL framework; that is, the context could instead refer to a set of closely-related states and similarly, actions can be endowed with a similarity measure under parameterization. For example, each input $\mathbf{x}_i$ in Ω consists of state features, a subset of which is used as action parameters under the parametric action model mentioned in Chapter 4. Since there is no limit imposed on how utility values could vary according to the state-action pairs in the decision process, we need to find a functional representation that is adaptive to the training samples and in the meantime, equipped with higher generalization capacity in order to capture interesting structures for the generalized state-action pairs.

5.2 Regression with Standard Linear Model

We will begin the derivation of a standard linear model by considering the training set $\Omega : \{(\mathbf{x}_1, q_1),...,(\mathbf{x}_m, q_m)\}$ mentioned earlier. A function in the standard linear regression model with independent and identically distributed (i.i.d) Gaussian noise can be expressed as follows:

$$Q_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = w_0 + w_1x_1 + w_2x_2 \ldots + w_nx_n,$$

$$q_i = Q_{\mathbf{w}}(\mathbf{x}_i) + \varepsilon_i, \tag{5.2}$$

where we have dropped the bias term b from (5.1) for convenience since the bias can simply be considered as a weight for the feature vector $\mathbf{x}$ with an additional component 1; namely, $\mathbf{x} = (1, x_1, x_2, ...)$ as an augmented vector. $Q_{\mathbf{w}}(\mathbf{x}_i)$ is the true function value to be estimated while q_i is the observed value in the training set. It may be helpful to imagine $Q_{\mathbf{w}}(\cdot)$ as a value function we wish to learn in an RL algorithm with $\mathbf{x}_i$ being the state feature vector and q_i being the utility. Moreover, additive noise ε is included here to model noisy observations. Assume that ε is a zero-mean Gaussian random variable with variance σ^2; i.e. $\varepsilon \sim N(0, \sigma^2)$. Thus, we can write

$$p(q_i \mid \mathbf{x}_i, \mathbf{w}, \sigma^2) = N(q_i \mid Q_{\mathbf{w}}(\mathbf{x}_i), \sigma^2), \tag{5.3}$$

such that q_i is predicted in this model in terms of a Gaussian distribution with a mean $Q_{\mathbf{w}}(\mathbf{x}_i)$ and variance σ^2.

Further, making the simplifying assumption that inputs $\{\mathbf{x}_i\} \subseteq X$ are drawn independently from the distribution (5.3), we obtain the following expression for the likelihood function (i.e. probability density of the observations given the parameters):

$$\begin{aligned} p(\mathbf{q} \mid X, \mathbf{w}, \sigma^2) &= \prod_{i=1}^{m} p(q_i \mid \mathbf{x}_i, \mathbf{w}, \sigma^2) \\ &= \prod_{i=1}^{m} N(q_i \mid Q_{\mathbf{w}}(\mathbf{x}_i), \sigma^2) \\ &= \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(q_i - \mathbf{x}_i^T \mathbf{w})^2}{2\sigma^2}\right) \end{aligned} \tag{5.4}$$

Taking the logarithm of the likelihood function (5.4), we have

$$\ln p(\mathbf{q} \mid X, \mathbf{w}, \sigma^2) = \sum_{i=1}^{m} \ln N(q_i \mid Q_{\mathbf{w}}(\mathbf{x}_i), \sigma^2)$$

$$= -\frac{m}{2}\ln\sigma^2 - \frac{m}{2}\ln(2\pi) - \frac{1}{\sigma^2}E_X(\mathbf{w}), \tag{5.5}$$

In (5.5), $E_X(\mathbf{w})$ defines a sum-of-squares error function:

$$E_X(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{m}\left[q_i - \mathbf{x}_i^T\mathbf{w}\right]^2 \tag{5.6}$$

With the likelihood function given by (5.5), we can apply maximum likelihood estimation (MLE) to determine **w**, which essentially maximizes the probability of the data. Notice that maximizing (5.5) is equivalent to minimizing the sum-of-squares error function $E_X(\mathbf{w})$. By taking the gradient with respect to $\mathbf{w}$ and set it to zero, i.e. $\nabla_{\mathbf{w}}E_X(\mathbf{w}) = 0$, we have:

$$\frac{\partial E}{\partial \mathbf{w}} = -X^T\mathbf{q} + X^T X\mathbf{w} = 0, \tag{5.7}$$

which yields the well-known normal equations:

$$X^T X\mathbf{w} = X^T\mathbf{q} \tag{5.8}$$

Thus, if the inverse of $X^T X$ exists, the solution to **w** is given by:

$$\mathbf{w} = (X^T X)^{-1}X^T\mathbf{q} \tag{5.9}$$

With the weight vector **w** determined, one can make a prediction over a new input in the following form:

$$\begin{aligned} Q_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} &= \mathbf{x}^T(X^T X)^{-1}X^T\mathbf{q}, \\ &= \sum_{i=1}^{m}\mathbf{x}^T(X^T X)^{-1}\mathbf{x}_i q_i \\ &= \sum_{i=1}^{m}k(\mathbf{x},\mathbf{x}_i)q_i, \\ &= \mathbf{k}^T\mathbf{q}, \end{aligned} \tag{5.10}$$

where $\mathbf{k}$ is a column vector with components in the form of inner products: $k(\mathbf{x},\mathbf{x}_i) = \mathbf{x}^T (X^T X)^{-1} \mathbf{x}_i$. In (5.10), we have also used the fact that $\mathbf{w}^T \mathbf{x} \in \mathbb{R}$, therefore, $\mathbf{w}^T \mathbf{x} = (\mathbf{w}^T \mathbf{x})^T = \mathbf{x}^T \mathbf{w}$. This gives us the first example indicating that the hypothesis function can be expressed through the inner product between the test point and the training sample. Note that $k(\mathbf{x},\mathbf{x}_i)$ in (5.10) can be thought of as computing a Mahalanobis distance [107], Δ, of the form:

$$\Delta^2 = (\mathbf{x} - \boldsymbol{\mu})^T \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu}),$$

where $\boldsymbol{\mu}$ comes from the same vector space as $\mathbf{x}$; i.e. $\boldsymbol{\mu} \in \mathbb{R}^n$. If Σ is an identity matrix, then Δ is reduced to an Euclidean distance from $\boldsymbol{\mu}$ to $\mathbf{x}$; on the other hand, (5.10) effectively computes such distance from the origin to $\mathbf{x}$ with Σ taken to be $X^T X$.

5.2.1 Basis Functions

The linear model like (5.10) suffers from limited expressiveness and in general will not be able to sufficiently express interesting value functions in realistic RL domains. One way to extend the model is to project the input into certain high dimensional space by taking a set of basis functions over the input to give: $\{\phi(\mathbf{x}_i) \mid i = 1,...,m\}$. Suppose that each input data is represented by a single variable x, , then the mapping $\phi(x) = (1, x, x^2, x^3, ...)^T$ projects the variable into a space of powers, i.e. $\phi_j(x) = x^j$. This can be extended to a higher dimensional input vector $\mathbf{x}$ by taking a dot product with a given vector $\mathbf{c}$ (e.g. a weight vector) before feeding it into the basis function; i.e. $\phi_j(\mathbf{x}) = \left(\langle \mathbf{c}, \mathbf{x} \rangle\right)^j$. Alternatively, the input $\mathbf{x}$ can be interpreted as having monomial features; i.e. $\phi_j(\mathbf{x}) = \prod_i x_i^{m_{ji}}$ where m_{ji} is a vector component for the *jth* basis function, specifying the exponent for the *ith* dimension of the input. One limitation of polynomial basis functions is that they are global functions of the input, and hence

any variations within the input will "broadcast" into other regions as well. Other possible choices of basis functions include, for instance, a square-exponential (SE) basis function that takes the following form:

$$\phi_j(\mathbf{x}) = \exp\left[-\frac{\left(\mathbf{x}-\mathbf{u}_j\right)^T\left(\mathbf{x}-\mathbf{u}_j\right)}{2l^2}\right] \tag{5.11}$$

In (5.11), l is a length-scale parameter, assuming that all input dimensions share the same length scale; $\mathbf{u}_j$ is a constant vector that serves as the center for the basis. On the other hand, Fourier basis in general takes the following form:

$$\phi_j(\mathbf{x}) = \left\{\cos\left(\pi\mathbf{c}_j \cdot \mathbf{x}\right), \sin(\pi\mathbf{c}_j \cdot \mathbf{x})\right\}, \tag{5.12}$$

where $\mathbf{x}, \mathbf{c}_j \in \mathbb{R}^n$, and leads to an expansion in the sinusoidal functions centered at various frequencies. In contrast to the radial basis function such as the SE basis in (5.11), each Fourier basis has an infinite spatial extent; SE basis, however, is localized to a finite region near its center (i.e. $\mathbf{u}_j$). In many signal processing applications, it is of interest to consider basis functions localized both in space and frequency, leading to a class of functions known as wavelets [105]. Another route for obtaining appropriate basis functions is to infer them from the topology implicitly defined by the training data. For instance, the *proto-reinforcement learning* framework [5, 6] represents the state space through samples of experience that encodes a graph, from which a graph Laplacian [26] is defined as a self-adjoined Laplace operator over the space of functions. The corresponding basis set is obtained as the eigenfunction of the Laplacian operator via spectral analysis. Another closely-related Laplacian approach uses *diffusion wavelets* [106] instead as the basis set, also a derivative from the graph Laplacian, for approximating the value function [3]. On a relevant note, we shall see in Chapter 7 an application of the graph Laplacian in the context of spectral clustering.

For the purpose of a general discussion, we will not specify a particular form of the basis function here. Substituting $\mathbf{x}$ for any basis function $\phi_j(\mathbf{x})$ in (5.2) gives,

$$Q_{\mathbf{w}}(\mathbf{x}) = \sum_{i=1}^{m} w_j \phi_j(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) \tag{5.13}$$

As a linear combination of non-linear basis functions, $Q_{\mathbf{w}}(\mathbf{x})$ now becomes non-linear but yet still remains linear with respect to the weight, giving rise to a linear smoother. Note the difference with the linear model (5.2), by which the prediction is expressed as a linear combination of the "raw input." Using (5.13), the minimization of the sum-of-squares error function in (5.7) then becomes:

$$\frac{\partial E_X}{\partial \mathbf{w}} = \sum_{i=1}^{m} \left[q_i - \mathbf{w}^T \phi(\mathbf{x}_i) \right] \phi(\mathbf{x}_i)^T = 0 \tag{5.14}$$

Rearranging the terms in (5.14) gives

$$\sum_{i=1}^{m} q_i \phi(\mathbf{x}_i)^T - \mathbf{w}^T \left(\sum_{i=1}^{m} \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^T \right) = 0 \tag{5.15}$$

Now, solving for **w**, we obtain a linear-smoother analogy for (5.9):

$$\mathbf{w} = \left(\Phi^T \Phi \right)^{-1} \Phi^T \mathbf{q} \tag{5.16}$$

Substituting (5.13) for **w** by (5.16), the prediction for a new input is thus given by:

$$\begin{aligned} Q_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) &= \phi(\mathbf{x})^T \left(\Phi^T \Phi \right)^{-1} \Phi^T \mathbf{q}, \\ &= \sum_{i=1}^{m} \phi(\mathbf{x})^T \left(\Phi^T \Phi \right)^{-1} \phi(\mathbf{x}_i) q_i \\ &= \sum_{i=1}^{m} k(\mathbf{x}, \mathbf{x}_i) q_i \\ &= \mathbf{k}^T \mathbf{q}, \end{aligned} \tag{5.17}$$

where in (5.17), $\mathbf{k}$ is effectively a generalization over the column vector in (5.10) with a notational overlap to indicate their similarity in terms of their functional form. However, with the basis $\phi(\cdot)$, $k(\mathbf{x},\mathbf{x}_i)$ is now generalized into to an inner product in the following form:

$$k(\mathbf{x},\mathbf{x}_i) = \langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle = \phi(\mathbf{x})^T \left(\Phi^T \Phi \right)^{-1} \phi(\mathbf{x}_i)$$

As can be seen from (5.17), the resulting hypothesis function $Q_{\mathbf{w}}(\cdot)$ again can be expressed in the form of a linear combination of inner products but this time, the inner product is evaluated in the space induced by the basis function map. Note that we have also introduced the notation $\langle \cdot,\cdot \rangle$ not only to draw an analogy to the dot product notation for vectors in $\mathbb{R}^n$ but also to begin to lay out the necessary setup for introducing the kernel function, which essentially computes an inner product in a more general sense – the notion of the inner product can be generalized to apply in complex vectors and complex functions (as infinite dimensional vectors). In fact, (5.17) also hints at the concept of the *kernel trick* [22]. For the moment, one can think of applying the kernel trick as replacing the basis function with another, which in turn induces a different inner product definition but yet shares the same functional structure in representing $Q_{\mathbf{w}}(\mathbf{x})$. In particular, notice that in order to compute $Q_{\mathbf{w}}(\mathbf{x})$ using (5.17), it is not necessary to know the explicit form of the basis function map $\phi(\cdot)$ provided that $k(\cdot,\cdot)$ is known. For reasons which will become clear shortly, in order to approximate the true function (e.g. potential value function in RL methods), making assumptions directly on the functional space through the kernel is often more intuitive than over the weight space using basis functions.

5.2.2 Dual Representation, Ridge Regression and Bayesian Regression Model

Both the linear model and the linear smoother given above lead to a similar representation for the function that interpolates a given training set in terms of a linear combination of inner products. Next, we shall apply the Bayesian treatment for the regression problem, which will again lead to a very similar representation for the function to be

approximated and draw the connection to Gaussian process regression (GPR) to be used in the upcoming chapters. In particular, Bayesian linear regression can be considered as a generalization of the MLE method for setting the parameters (i.e. the weight vector $\mathbf{w}$ for $Q_{\mathbf{w}}(\mathbf{x})$) in a linear regression model by introducing an additional regularization term to the likelihood function (5.5). Effectively, this is to introduce an additional penalty term to the error function $E_X(\mathbf{w})$ in (5.6):

$$E_X(\mathbf{w}) + \lambda E_W(\mathbf{w}), \tag{5.18}$$

where $\lambda \in R^+$ is a scalar multiplier. One of the simplest regularizer is the sum-of-squares of the weight vector component:

$$E_W(\mathbf{w}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} \tag{5.19}$$

Thus, applying (5.6), (5.19) to (5.18) yields the total error function:

$$E_X(\mathbf{w}) + \lambda E_W(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{m}\left[q_i - \mathbf{x}_i^T\mathbf{w}\right]^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w} \tag{5.20}$$

The introduction of the regularization term $E_W(\mathbf{w})$ effectively controls the model complexity of the hypothesis function $Q_{\mathbf{w}}(\mathbf{x})$ by disallowing excessively large weights in order to prevent from overfitting of the data. In other words, minimizing the total error in (5.20) will encourage the weight value to decay towards zero unless otherwise supported by the training set.

Minimizing the error function with a regularizer also has its connection to *ridge regression* [1, 107]. Recall from the normal equation (5.8) (and similarly (5.16)) that, in order to obtain the solution for the weight vector $\mathbf{w}$, the data matrix X^TX has to be invertible. If instead X^TX is not of full rank or if numerical stability problems occur in the matrix inversion, then one can use the following solution:

$$\mathbf{w} = (X^TX + \lambda I_m)^{-1}X^T\mathbf{q}, \tag{5.21}$$

where in (5.21), λ corresponds to the scalar multiplier in the total error function (5.20) and I_m is an *m-by-m* identify matrix. To see the connection between the regularizer $\lambda E_W(\mathbf{w})$ and the solution for $\mathbf{w}$ by ridge regression, i.e. (5.21), let $L(\mathbf{w}) = E_X(\mathbf{w}) + \lambda E_W(\mathbf{w})$ to give:

$$L(\mathbf{w}) = \frac{\lambda}{2}\langle \mathbf{w}, \mathbf{w} \rangle + \frac{1}{2}\sum_{n=1}^{m}\left[q_i - \mathbf{x}_i^T \mathbf{w}\right]^2, \tag{5.22}$$

Minimizing (5.22) with respect to the weight vector $\mathbf{w}$ by setting $\partial L(\mathbf{w}) / \partial \mathbf{w}$ to zero, we see that the solution for $\mathbf{w}$ takes the following form:

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{i=1}^{m}(\mathbf{x}_i^T \mathbf{w} - q_i)\mathbf{x}_i = \sum_{i=1}^{m}\alpha_i \mathbf{x}_i, \tag{5.23}$$

where α_i is defined as $-\frac{1}{\lambda}(\mathbf{x}_i^T \mathbf{w} - q_i)$ (also compare to (5.1)). We now work with the parameter $\boldsymbol{\alpha}$ instead of the original weight vector $\mathbf{w}$ by reformulating the error function in terms of $\boldsymbol{\alpha}$. Doing so leads to an alternative expression known as the dual representation for $Q_{\mathbf{w}}(\mathbf{x})$ (as opposed to the representation in the original weight space). Consequently, we can derive conditions that $\boldsymbol{\alpha}$ must satisfy. Specifically, rewriting (5.22) using (5.23) by setting $\mathbf{w} = X^T \boldsymbol{\alpha}$ and $K = XX^T$ gives:

$$\begin{aligned}
\frac{1}{2}L(\boldsymbol{\alpha}) &= \lambda \boldsymbol{\alpha}^T XX^T \boldsymbol{\alpha} + \sum_{i=1}^{m}\left(\boldsymbol{\alpha}^T X\mathbf{x}_i - q_i\right)^2 \\
&= \lambda \boldsymbol{\alpha}^T K\boldsymbol{\alpha} + \sum_{i=1}^{m}\left((K\boldsymbol{\alpha})_i - q_i\right)^2 \\
&= \lambda \boldsymbol{\alpha}^T K\boldsymbol{\alpha} + \left(K\boldsymbol{\alpha} - \mathbf{q}\right)^T\left(K\boldsymbol{\alpha} - \mathbf{q}\right) \\
&= \lambda \boldsymbol{\alpha}^T K\boldsymbol{\alpha} + \boldsymbol{\alpha}^T KK\boldsymbol{\alpha} - 2\mathbf{q}^T K\boldsymbol{\alpha} + \mathbf{q}^T\mathbf{q}
\end{aligned} \tag{5.24}$$

Setting $\partial L(\boldsymbol{\alpha}) / \partial \boldsymbol{\alpha}$ to zero, we get

$$\left(\lambda I + K\right)\boldsymbol{\alpha} = \mathbf{q} \tag{5.25}$$

Thus, analogous to (5.10) but with a regularization term, the hypothesis function can be written as:

$$Q_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \mathbf{q}^T(\lambda I + XX^T)^{-1}X^T\mathbf{x} = \mathbf{q}^T(\lambda I + K)^{-1}\mathbf{k}, \tag{5.26}$$

where **k** again represents the column vector with each component being an inner product $\langle \mathbf{x}, \mathbf{x}_i \rangle$ although in a slightly different form than (5.10). Similarly, in the case of the hypothesis function represented as a linear combination of basis functions by mapping **x** to $\phi(\mathbf{x})$, the corresponding $Q_{\mathbf{w}}(\mathbf{x})$ in (5.17) then becomes:

$$Q_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\phi(\mathbf{x}) = \mathbf{q}^T(\lambda I + \Phi\Phi^T)^{-1}\mathbf{k}, \tag{5.27}$$

by which we would like to reiterate that $k(\mathbf{x}, \mathbf{x}_i)$ corresponds to $\langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle$ as the *ith* component of **k** in (5.27).

Further, the dual representation also has its trace in optimization methods that approach the solution by minimizing an error function. Specifically, one can view the total error as being contributed individually by members of the training set. That is, each displacement between the true function value and the hypothesis is represented by a slack variable $\xi_i = q_i - \langle \mathbf{w}, \mathbf{x}_i \rangle$. Thus minimizing (5.22) can be formulated as a constraint optimization problem:

$$\begin{aligned} &\text{minimize} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{m}\xi_i^2, \\ &\text{subject to } \xi_i = q_i - \langle \mathbf{w}, \mathbf{x}_i \rangle, \text{ where } i = 1,...,m \end{aligned} \tag{5.28}$$

From (5.28), we derive the following Lagrangian:

$$L(\mathbf{w}) = \frac{\lambda}{2}\|\mathbf{w}\|^2 + \sum_{i=1}^{m}\xi_i^2 + \sum_{i=1}^{m}\alpha_i\left(q_i - \langle \mathbf{w}, \mathbf{x}_i \rangle - \xi_i\right), \tag{5.29}$$

Minimizing (5.29) by differentiating and imposing a stationarity condition leads to the same solution for $\mathbf{w}$ in terms of $\boldsymbol{\alpha}$; that is, $\mathbf{w} = \frac{1}{\lambda}\sum_{i=1}^{m}\alpha_i \mathbf{x}_i$ and $\xi_i = \frac{\alpha_i}{2}$. Substituting $\mathbf{w}$ and ξ_i in (5.29) and reformulating $L(\mathbf{w})$ in terms of $\boldsymbol{\alpha}$, we get the following dual problem:

$$W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} q_i \alpha_i - \frac{1}{2\lambda}\sum_{i,j=1}^{m}\alpha_i \alpha_j \left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle - \frac{1}{4}\sum_{i=1}^{m}\alpha_i \tag{5.30}$$

For the convenience of derivation, we can rewrite (5.30) in matrix form to get,

$$W(\boldsymbol{\alpha}) = \mathbf{q}^T\boldsymbol{\alpha} - \frac{1}{2\lambda}\boldsymbol{\alpha}^T K \boldsymbol{\alpha} - \frac{1}{4}\boldsymbol{\alpha}^T\boldsymbol{\alpha} \tag{5.31}$$

where K represents a Gram matrix (see Section 5.3) with entries $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \left\langle \mathbf{x}_i, \mathbf{x}_j \right\rangle$. Next, maximizing (5.31) by setting $\partial W(\boldsymbol{\alpha}) / \partial \boldsymbol{\alpha} = 0$ leads to:

$$\boldsymbol{\alpha} = \lambda(K + \lambda I)^{-1}\mathbf{q} \tag{5.32}$$

Using the relation $\mathbf{w} = \frac{1}{\lambda}X^T\boldsymbol{\alpha}$, we obtain the exact same result derived from minimizing the error function directly, i.e. $Q_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \mathbf{q}^T(\lambda I + K)^{-1}\mathbf{k}$. Thus, from the perspective of (5.28) and (5.30), we see that the dual representation in terms of $\boldsymbol{\alpha}$ has its root in the dual optimization problem in which minimizing the Lagrangian $L(\mathbf{w})$ corresponds to maximizing the dual problem $W(\boldsymbol{\alpha})$. In fact, this is how the "regular" ridge regression is extended to *kernel ridge regression* in which the function $k(\cdot,\cdot)$ used to compute the matrix K in (5.32) can be replaced by any valid kernels – another manifestation of the kernel trick. The dual representation of the hypothesis function $Q_{\mathbf{w}}(\mathbf{x})$ derived from ridge regression or from the optimization perspective can also be obtained by directly working through the Bayesian regression approach, which also operates in the weight space (i.e. representing $Q_{\mathbf{w}}(\mathbf{x})$ in terms of a proper weight vector $\mathbf{w}$). In the Bayesian regression formulation, a prior probability distribution over the weight is specified.

Suppose that we put a Gaussian prior with mean $\mathbf{m}$ and covariance matrix Σ over weights, this gives:

$$p(\mathbf{w}) = N(\mathbf{m}, \Sigma) \tag{5.33}$$

Inference in the Bayesian regression model is based on the posterior distribution over weights using Bayes' rule:

$$p(\mathbf{w} \mid \mathbf{q}, X) = \frac{p(\mathbf{q} \mid X, \mathbf{w}) p(\mathbf{w})}{p(\mathbf{q} \mid X)}, \tag{5.34}$$

where the denominator $p(\mathbf{q} \mid X)$ is known as the marginal likelihood obtained by integrating out (or marginalizing over) the weight; i.e.

$$p(\mathbf{q} \mid X) = \int p(\mathbf{q} \mid X, \mathbf{w}) p(\mathbf{w}) \, d\mathbf{w} \tag{5.35}$$

The posterior weight in (5.34) combines the prior (5.33) and the likelihood function (5.4), capturing everything known about the weight parameter. In addition, due to the choice of a conjugate Gaussian prior distribution, the posterior will also be Gaussian. For simplicity, assume that the Gaussian prior has zero mean; i.e. $\mathbf{m} = \mathbf{0}$. Here, we shall directly consider the case where $Q_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x})$ by using the basis set ϕ. Writing only the term that depends on weights including the likelihood function and the prior, and subsequently performing the usual procedure of completing the square, we get:

$$\begin{aligned} p(\mathbf{w} \mid X, \mathbf{q}, \sigma^2) &\propto \exp\left[-\frac{1}{2\sigma^2} \left(\mathbf{q} - \Phi^T \mathbf{w} \right)^T \left(\mathbf{q} - \Phi^T \mathbf{w} \right) \right] \exp\left(-\frac{1}{2} \mathbf{w}^T \sum \mathbf{w} \right) \\ &\propto \exp\left[-\frac{1}{2} (\mathbf{w} - \bar{\mathbf{w}})^T \left(\frac{1}{\sigma^2} \Phi^T \Phi + \sum^{-1} \right) (\mathbf{w} - \bar{\mathbf{w}}) \right], \end{aligned} \tag{5.36}$$

where $\bar{\mathbf{w}} = \frac{1}{\sigma^2} A^{-1} \Phi^T \mathbf{q}$. From this, we recognize that the form of the probability density (5.36) corresponds to the Gaussian posterior distribution with mean $\bar{\mathbf{w}}$ and covariance matrix $A^{-1} = \left(\sigma^{-2} \Phi^T \Phi + \sum^{-1} \right)^{-1}$. To further simplify the derivation and drawing its connection back to

ridge regression mentioned earlier, consider a zero-mean isotropic Gaussian governed by a single precision parameter λ / σ^2 such that

$$p(\mathbf{w} \mid \lambda) = N(\mathbf{0}, \frac{\sigma^2}{\lambda} I), \tag{5.37}$$

where λ can be thought of as a control parameter that regularizes the precision (the reciprocal of the variance) in (5.37). The corresponding posterior distribution over **w** thus is reduced to the Gaussian with mean $\bar{\mathbf{w}} = \left(\Phi^T \Phi + \lambda I\right)^{-1} \Phi^T \mathbf{q}$ and covariance $A^{-1} = \sigma^2 \left(\Phi^T \Phi + \lambda I\right)^{-1}$. Given this, we can express the hypothesis function in terms of the mean prediction over the weight vector:

$$\begin{aligned} Q_{\mathbf{w}}(\mathbf{x}) = \bar{\mathbf{w}}^T \phi(\mathbf{x}) &= \phi(\mathbf{x})^T (\Phi^T \Phi + \lambda I)^{-1} \Phi^T \mathbf{q} \\ &= \sum_{i=1}^{m} \phi(\mathbf{x})^T (\Phi^T \Phi + \lambda I)^{-1} \phi(\mathbf{x}_i) q_i \\ &= \sum_{i=1}^{m} k(\mathbf{x}, \mathbf{x}_i)\, q_i \end{aligned} \tag{5.38}$$

In (5.38), we have again defined $k(\mathbf{x}, \mathbf{x}_i)$ to be $\phi(\mathbf{x})^T (\Phi^T \Phi + \lambda I)^{-1} \phi(\mathbf{x}_i)$ as a way of computing the inner product in the feature space induced by the basis function map. Note that the kernel defined in terms of (5.38) is data-dependent; that is, $k(\cdot, \mathbf{x}_i)$ can be thought of as a function that takes input data $\mathbf{x}$ as an argument and returns a value in a manner that depends on the known training data $\{\mathbf{x}_i\} \subseteq X$. The definition of $k(\cdot, \mathbf{x}_i)$ in terms of (5.38) is also known as the *smoother matrix* or the *equivalent kernel* (EK) [37, 107]. From (5.38), we see that the prediction of $Q_{\mathbf{w}}(\mathbf{x})$ using the average weight vector is identical to (5.17). Moreover, the predictive distribution can be obtained as an average over all possible linear models with respect to the Gaussian posterior:

$$p(q_* \mid X, \mathbf{q}) = \int p(q_* \mid X, \mathbf{w}) p(\mathbf{w} \mid X, \mathbf{q}) d\mathbf{w}$$

$$= N(\frac{1}{\sigma^2}\mathbf{x}_*^T A^{-1}\Phi\mathbf{q},\ \mathbf{x}_*^T A^{-1}\mathbf{x}_*), \tag{5.39}$$

where the mean of the predictive distribution is equivalent to (5.38) and the variance is a quadratic form of the test input $\mathbf{x}_*$ with the posterior covariance matrix A. Note that we have explicitly denoted the test input and its corresponding predictive output with an asterisk in the subscript in order to differentiate it from the old data; i.e. $(X,\mathbf{q})$. Notice that for the Bayesian regression model, the mean of the posterior $p(\mathbf{w} \mid X,\mathbf{q},\sigma^2)$ is also its mode, which is often referred to as the *maximum a posteriori* (MAP) estimate of the weight vector **w**. Further, the corresponding log likelihood function with respect to the posterior can be expressed as:

$$\begin{aligned}\ln p(\mathbf{q} \mid X,\mathbf{w},\sigma^2) &= \sum_{i=1}^{m} ln\, N(\bar{\mathbf{w}}_i{}^T \phi(\mathbf{x}_i), A^{-1}) \\ &= -\frac{1}{2\sigma^2}\sum_{n=1}^{m}\left[q_i - \mathbf{w}^T\phi(\mathbf{x}_i)\right]^2 - \frac{\lambda}{2\sigma^2}\mathbf{w}^T\mathbf{w} + const. \\ &= -\frac{1}{\sigma^2}\left[E_X(\mathbf{w}) + \lambda E_W(\mathbf{w})\right] + const \end{aligned} \tag{5.40}$$

Comparing (5.40) with (5.20), we see that maximizing this log likelihood with respect to **w** is equivalent to minimizing the sum-of-squares error $E_X(\mathbf{w})$ with an additional quadratic regularization term $\lambda E_W(\mathbf{w})$. Thus, the prior over weights in the Bayesian linear regression effectively serve as a regularizer. In particular, choosing the precision parameter (see (5.37)) for the prior corresponds to choosing an appropriate λ in (5.22), which in turn controls the complexity of the model by penalizing large weights during the minimization process of the total error function.

5.2.3 Limitations in Fixed Basis Functions

Although with an appropriate selection of basis functions, we can in principle model a function (e.g. the value function in RL) with arbitrary nonlinear properties, there are significant limitations in terms of generalization and adaptability to the training data. In particular, recall the task assignment domain from earlier chapters where each assignment action can lead to significantly different results due to the time-varying property of the computational resource. Expressing the control policy in complex domains as such therefore would require an adaptive value function representation that can be progressively updated in response to the varying system dynamics. Moreover, a value function represented by fixed basis functions may not be able to generalize the learned experience from the state space visited so far to the unknown parts of the state space. Specifically, recall the form of the hypothesis function in (5.38), where the averaged weight obtained from the posterior distribution effectively determines how the value of any future input will be estimated. Since the prediction is achieved through a linear combination of training samples with these weights, any prediction using localized basis functions (e.g. the SE basis in (5.11)) that falls outside the range of the known samples will be effectively zero. This is due to the fact that localized basis functions "centered" around the known training samples will eventually fade away in magnitude. As a consequence, a linear combination of these basis functions outside their effective range will evaluate to approximately zero under any possible weight distribution. Yet, in order to develop a robust learning framework, it is desirable to nonetheless obtain reasonable value predictions over the unknown parts of the state space in that the nature of policy learning in the RL framework hinges upon a progressive advancement in the unknown state space.

In addition, the difficulty in applying a fixed set of basis functions in the RL framework is also rooted deep down in the representation of standard formulations in which the state space grows exponentially with a linear expansion of features. Therefore, the size of the basis set can still grow rapidly with the dimension of the state space. A dynamic increase in the state space

occurs, for instance, in the task assignment problem for the generic workload management system introduced in Chapter 2. In particular, as the pilots are distributed across the resource-sharing environment, the set of effective computational resources can grow rapidly, which implies that the state features necessary to characterize the candidate machine will increase accordingly.

Challenges also arise in choosing the right basis set that, if sufficiently large, generalizes well enough to reflect the topology of the state space. It is often unclear how the right size of the basis set can be determined along with the proper functional form without some experience of trial and error. In addition, to learn a policy for the task assignment in a versatile set of candidate machines with non-stationary resource capacity, the policy representation needs to be adjustable to reflect a time-shifting resource profile. Accordingly, we need a value function approximator that enables a dynamic tuning of the underlying basis function in order to reestablish the accuracy of its future prediction without a substantial re-learning process. However, the intuition gained from the ridge regression and the Bayesian regression approach does serve as pointers to the desired representation we are seeking. This will become clearer as we introduce the notion of kernel functions, which already had their traces in the regression methods introduced earlier.

5.3 Kernel

Kernel representations allow for a generalization of the dot product into Hilbert space (also referred to as inner product) where more complex functions can be expressed through a generalized dot product (i.e. kernel trick [22]). In particular, we will study the class of kernels [1, 22] that correspond to inner products in feature space H via a map $\Phi : X \to H$. That is, a sample $\mathbf{x}$ is represented in H through the following mappings:

$$\mathbf{x} = (x_1, x_2, ..., x_N) \mapsto \Phi(\mathbf{x}) = \left(\phi_1(\mathbf{x}), ..., \phi_M(\mathbf{x})\right)$$

Definition 5.1 (**Kernel**) [1] *A kernel is a function* k, *such that for all* $\mathbf{x}, \mathbf{x}' \in X$

$$k(\mathbf{x}, \mathbf{x}') = \left\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \right\rangle$$

where Φ *is a map from* X *to the feature space H endowed with inner products*.

In order to represent useful and practical similarity measures, it is necessary to ensure that a kernel function $k(\mathbf{x}, \mathbf{x}')$ is indeed a valid kernel and thus has certain properties. First, the function must be symmetric:

$$k(\mathbf{x}, \mathbf{x}') = \left\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \right\rangle = \left\langle \Phi(\mathbf{x}'), \Phi(\mathbf{x}) \right\rangle = k(\mathbf{x}', \mathbf{x}) \tag{5.41}$$

Next, it must follow the Cauchy-Schwarz inequality in order to allow to compare two input vectors in terms of the dot product (which implicitly defines the angle, triangular inequality in feature space, etc):

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}')^2 &= \left\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \right\rangle^2 \\ &\leq \left\| \Phi(\mathbf{x}) \right\|^2 \left\| \Phi(\mathbf{x}') \right\|^2 \\ &= \left\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}) \right\rangle \left\langle \Phi(\mathbf{x}'), \Phi(\mathbf{x}') \right\rangle \\ &= k(\mathbf{x}, \mathbf{x}) k(\mathbf{x}', \mathbf{x}') \end{aligned} \tag{5.42}$$

Additionally, in order to guarantee the existence of a feature space induced by the kernel, we need to ensure that any vector drawn from the feature space assumes a norm that is non-negative and real-valued (otherwise, it would contradict the geometry of that space and hence the kernel would not serve as a useful similarity measure). Intuitively, this can be achieved by noting that any point in the feature space can be represented by a linear combination of the feature map $\{\phi_i \mid i = 1, ...M\}$. Recall that a given input $\mathbf{x}$ is represented by the sequence $\left(\phi_1(\mathbf{x}), ..., \phi_M(\mathbf{x})\right)$ through the map $\Phi(\cdot)$. Thus with n inputs, we have n corresponding points in

the feature space represented by the set: $\left\{\left\langle \phi_1(\mathbf{x}_i),...,\phi_M(\mathbf{x}_i)\right\rangle \mid i=1,...n\right\}$, each of which stretches *M* dimensions. Hence, for an arbitrary point represented as $\sum_i c_i\Phi(\mathbf{x}_i)$, where $c_i \in \mathbb{R}$, the norm is given by:

$$\left\|\sum_i c_i\Phi(\mathbf{x}_i)\right\|^2 = \left\langle \sum_i c_i\Phi(\mathbf{x}_i), \sum_j c_j\Phi(\mathbf{x}_j)\right\rangle$$

$$= \sum_{i,j} c_i c_j k(\mathbf{x}_i,\mathbf{x}_j), \tag{5.43}$$

and with the non-negativity of the norm, we have $\sum_{i,j} c_i c_j k(\mathbf{x}_i,\mathbf{x}_j) \geq 0$. Note that the first equality in (5.43) follows from the assumption that the dot product $\langle\cdot,\cdot\rangle$ is bilinear. The following proposition includes the requirements of all the desired kernel properties described above:

Proposition 5.2 [22] Let *X be a finite input space with* $k(\mathbf{x},\mathbf{x}')$ *being a symmetric function on X. Then* $k(\mathbf{x},\mathbf{x}')$ *is a kernel function if and only if the (Gram) matrix K with entries:*

$$K_{ij} = k(\mathbf{x}_i,\mathbf{x}_j)$$

is positive semi-definite (PSD); hence, with non-negative eigenvalues.

Alternatively, one can say a real symmetric *n-by-n* matrix *K* satisfying $\sum_{i,j} c_i c_j K_{ij} \geq 0$ or equivalently, $\mathbf{c}^T K\mathbf{c} \geq 0$ for all $\mathbf{c} \in \mathbb{R}^n$, is positive semi-definite. Notice that, the positive (semi-)definiteness automatically implies the symmetry, $k(\mathbf{x},\mathbf{x}') = k(\mathbf{x}',\mathbf{x})$, mentioned earlier and the Cauchy-Schwarz inequality for kernels, $\left\|k(\mathbf{x},\mathbf{x}')\right\|^2 \leq k(\mathbf{x},\mathbf{x}')\cdot k(\mathbf{x}',\mathbf{x})$, both being desired kernel properties. As we shall see shortly, the PSD kernel determined through the condition of the form: $\mathbf{c}^T K\mathbf{c} \geq 0$ is a finite-dimensional analogy to Mercer's Theorem. Mercer's Theorem provides a more general characterization of the property that has to hold for a valid kernel.

5.3.1 Mercer's Theorem

The term kernel stems from the theory of integral operators, where the operator T_k is defined as [22]:

$$\left(T_k f\right)(\mathbf{x}) = \int_X k(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') d\mu(\mathbf{x}'), \tag{5.44}$$

where μ denotes a measure. A practical use case in the context of the RL framework would be to consider μ as a probability measure (e.g. probability over state distribution), in which case, readers will usually be able to substitute $p(\mathbf{x})d\mathbf{x}$ for $d\mu(\mathbf{x})$ assuming that the probability density $p(\mathbf{x})$ exists. Effectively, T_k takes on an input in the form of a function $f \in L_2(X)$ and maps it to another function also in $L_2(X)$. Drawing an analogy to the finite dimensional case, (5.44) corresponds to a (symmetric) matrix operating on a column vector with each component obtained by sampling a mesh of points $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m\}$ over the function $f(\cdot)$ by multiplying with a shifted delta function, i.e. $\sum_{i=1}^{m} \delta(\mathbf{x} - \mathbf{x}_i)$, to obtain corresponding sampled targets $\{f(\mathbf{x}_1), f(\mathbf{x}_2), ..., f(\mathbf{x}_m)\}$.

As we shall see later on, Gaussian process regression [14, 107] can be viewed as Bayesian linear regression with a possibly infinite number of basis functions, which is an important property that motivates Gaussian process regression as a mechanism for value predictions. One possible basis set is the eigenfunction of the kernel. A function $\phi(\cdot)$ that satisfies the integral equation:

$$\int k(\mathbf{x}, \mathbf{x}') \phi(\mathbf{x}) d\mu(\mathbf{x}) = \lambda \phi(\mathbf{x}'), \tag{5.45}$$

is called an eigenfunction of kernel k with corresponding eigenvalue λ, all evaluated with respective to the measure μ. Similar to a counterpart in the finite case where a symmetric matrix has an eigen-decomposition with eigenvectors that can be chosen to be orthonormal, here the eigenfunction also follow analogous properties. That is, eigenfunctions of the kernel k

are orthogonal with respect to μ and can be chosen to be normalized so that $\int \phi_i(\mathbf{x})\phi_j(\mathbf{x})d\mu(\mathbf{x}) = \delta_{ij}$ where δ_{ij} is the Kronecker delta function; i.e. $\delta_{ij} = 1$ if $i = j$ and $\delta_{ij} = 0, \text{otherwise}$.

While further analysis of the integral operator is out of the scope of this dissertation, we shall quote the theorem for defining a valid kernel, based on which to gain intuition for the desired properties inherent in a Gaussian-process utility predictor to be discussed in the next chapter.

Theorem 5.3 (**Mercer's Theorem**) [1, 22] *Let X be a compact subset of* $\mathbb{R}^n$. *Suppose k is a symmetric real-valued function such that the integral operator* $T_k : L_2(X) \to L_2(X)$ *and*

$$(T_k f)(\mathbf{x}) = \int_X k(\mathbf{x},\mathbf{x}')f(\mathbf{x}')d\mu(\mathbf{x}')$$

is non-negative, that is for all $f \in L_2(X)$, *we have*

$$\int_{X\times X} k(\mathbf{x},\mathbf{x}')f(\mathbf{x})f(\mathbf{x}')\,d\mu(\mathbf{x})d\mu(\mathbf{x}') \geq 0$$

Let $\phi_j \in L_2(X)$ *be the normalized orthogonal eigenfunctions of* T_k *(i.e.* $\|\phi_j\|_{L_2} = 1$*) associated with the eigenvalues* λ_j *in a non-decreasing order. Then,* $k(\mathbf{x},\mathbf{x}')$ *can be expanded in terms of the eigenfunctions* $\{\phi_j\}$ *as follows:*

$$k(\mathbf{x},\mathbf{x}') = \sum_{j=1}^{N_H} \lambda_j \phi_j(\mathbf{x})\phi_j(\mathbf{x}')$$

where N_H *can be either finite:* $N_H \in \mathbb{N}$ *or infinite:* $N_H \to \infty$; *in the latter case, the series converges absolutely and uniformly on* $X \times X$.

Applying a sampling process with a shifted delta function over the function $f(\cdot)$ in Mercer's theorem and extracting its outputs to form a finite dimensional vector **v**, we see that the eigen-decomposition of $k(\cdot,\cdot)$ reduces to the diagonalization of a symmetric matrix. In

addition, the kernel assuming the representation of eigenfunction expansion above together with the form of the positive condition in Mercer's Theorem

$$\int_{X\times X} k(\mathbf{x},\mathbf{x}')f(\mathbf{x})f(\mathbf{x}')\,d\mu(\mathbf{x})d\mu(\mathbf{x}') \geq 0, \tag{5.46}$$

is an infinite dimensional analogy to the positive semi-definite condition in the finite case (Proposition 5.2). To see this, again let f be the weighted sums of delta functions at each $\mathbf{x}_i$. Since such functions are limits of functions in $L_2(X)$, the above condition implies that for any finite subset of X, the corresponding Gram matrix K is always positive semi-definite.

In addition, the kernel function that expands into an infinite series (i.e. having infinite number of non-zero eigenvalues) is a *non-degenerative kernel* such as the squared exponential kernel,

$$k(\mathbf{x},\mathbf{x}') = \exp(-\frac{\left\|\mathbf{x}-\mathbf{x}'\right\|^2}{2l^2}),$$

where l is a length-scale parameter. We shall see a more general form of this kernel shortly. On the other hand, if the kernel has only a finite number of eigenvalues, thus expanding only into finite terms, then it is *degenerative*; for instance, a constant σ_0^2, a polynomial kernel $\left(\mathbf{x}\cdot\mathbf{x}'+\sigma_0^2\right)^p$, etc are all degenerative.

5.4 From Bayesian Linear Regression to Gaussian Process Regression

Bayesian linear regression introduced earlier formulates the hypothesis function from the weight space perspective. That is, the predictive output is expressed as an averaging over all possible linear models (in the form of Gaussians) through posterior weights. The predictive distribution is thus again Gaussian. Gaussian process regression (GPR), on the other hand, arrives at the desired hypothesis function from the space of functions. A Gaussian process is

effectively a probability distribution over a collection of random variables, any arbitrary set of which jointly has a Gaussian distribution.

A Gaussian process $Q(\mathbf{x}) \sim \mathcal{GP}\left(\bar{Q}(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\right)$ is completely specified by the second-order statistics; namely a mean function $\bar{Q}(\mathbf{x}) = \mathrm{E}[Q(\mathbf{x})]$ and a covariance function (a kernel),

$$k(\mathbf{x}, \mathbf{x}') = \mathrm{E}[(Q(\mathbf{x}) - \bar{Q}(\mathbf{x}))(Q(\mathbf{x}') - \bar{Q}(\mathbf{x}')].$$

Note that the function $Q(\mathbf{x})$ modeled by a GP is analogous to the hypothesis function given earlier: $Q_w(\mathbf{x})$. The difference is that $Q_w(\mathbf{x})$ was represented by a linear combination of basis functions with a corresponding weight vector w, giving rise to a weight space view of the hypothesis function. The function $Q(\mathbf{x})$ on the other hand is directly modeled from the space of functions whose properties are specified by the kernel $k(\cdot,\cdot)$. Note also that any positive-definite kernel can be used as a covariance function since such a kernel function induces a positive-definite Gram matrix (see Proposition 5.2), which is the necessary and sufficient condition for a valid covariance matrix. In Chapter 6, we shall see that there is an advantage in explicitly modeling a parametric action as a random vector in which each vector component follows a task-dependent probability distribution with values bounded by a finite set (i.e. a constrained random variable). With the formulation of "noisy actions," how do we still obtain reasonable value predictions for parametric actions with stochastic effects as they are applied in the state space? By associating the Bayesian liner regression model we have seen earlier and making a transition from the weight space hypothesis (i.e. making prior assumption over weights) to a function space hypothesis, we could first imagine a set of training data of the form:

$$\Omega' : \left\{(\mathbf{x}_1', f_1), (\mathbf{x}_2', f_2), ..., (\mathbf{x}_3', f_m)\right\} \subseteq X' \times F_Q,$$

where $\left\{\mathbf{x}_i'\right\} \subseteq X'$ denotes a set of noise-free observations of state-action pairs as vectors and $\left\{f_i'\right\} \subseteq F_Q$ corresponds to the set of reinforcement signals (e.g. utilities) of noise-free state-

action pairs. It suffices, at this point, to assume that each state-action pair is represented by a vector in which both the state and action have fixed values. In contrast to Ω', the functional data set mentioned earlier in the chapter,

$$\Omega : \{(\mathbf{x}_1, q_1), (\mathbf{x}_2, q_2), ..., (\mathbf{x}_3, f_m)\} \subseteq X \times \mathbf{Q},$$

can thus be used to represent a set of noisy state-action-pairs, each of which is mapped to a noisy scalar signal. In particular, with the data set Ω and Ω' given above, we can approach the functional-space model by adapting (5.35); namely, instead of marginalizing over weights, we marginalize over function values (target signals) to give:

$$p(\mathbf{q} \mid X) = \int p(\mathbf{q} \mid \mathbf{f}, X) p(\mathbf{f} \mid X) d\mathbf{f}, \tag{5.47}$$

Similar to the approach using a prior assumption over weights in the case of the Bayesian regression model (e.g. (5.33) and (5.37)), assuming that $\mathbf{f} \mid X \sim N(\mathbf{0}, K)$ leads to the log likelihood function is thus given by:

$$\log p(\mathbf{f} \mid X) = -\frac{1}{2}\mathbf{f}^T K^{-1}\mathbf{f} - \frac{1}{2}\log|K| - \frac{n}{2}\log 2\pi \tag{5.48}$$

Notice the similarity between (5.48) and (5.5) by substituting the weight-dependent error function $E_X(\mathbf{w})$ for (5.6) such that the log likelihood in (5.5) can be expressed solely in terms of the data and observed signals:

$$\ln p(\mathbf{q} \mid X, \mathbf{w}) = \frac{1}{2}\left(\frac{1}{\sigma^2}\sum_{n=1}^{m}\left[q_i - \mathbf{x}_i^T \mathbf{w}\right]^2\right) - \frac{m}{2}\ln\sigma^2 - \frac{m}{2}\ln(2\pi),$$

$$= -\frac{1}{2}\mathbf{w}^T \Sigma^{-1}\mathbf{w} - \frac{1}{2}\log|\Sigma| - \frac{n}{2}\log 2\pi, \tag{5.49}$$

where in (5.49), we have defined $\Sigma = \sigma^2 I$, by assuming that the variance of the input is σ^2 and that each input is drawn independently following the same Gaussian noise assumption. Note that in the weight-space model, the prior over weights are assumed to be independent of the

data X for simplicity and thus, $p(\mathbf{w} \mid X) = p(\mathbf{w}) = N(\mathbf{0}, \Sigma) = N(\mathbf{0}, \sigma^2 I)$. Conversely, the prior over function values is explicitly modeled by a covariance matrix K evaluated pair-wise from the data set X through the kernel; i.e. $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$. In the case where components of $\mathbf{q}$ correspond to noisy observations of $\mathbf{f}$, the log likelihood in (5.48) is shifted to the following:

$$\log p(\mathbf{q} \mid X) = -\frac{1}{2}\mathbf{q}^T \left(K + \sigma^2 I\right)^{-1} \mathbf{q} - \frac{1}{2}\log\left|K + \sigma^2 I\right| - \frac{n}{2}\log 2\pi \tag{5.50}$$

The result of (5.50) can be obtained directly by assuming $\mathbf{q} \mid X \sim N(\mathbf{0}, K + \sigma^2 I)$. Alternatively, one can use a noisy kernel to explicitly incorporate the noise assumption such that $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) + \sigma^2 \delta_{ij}$, where δ_{ij} is a Kronecker delta function. In this case, (5.50) is reduced to (5.48) by "absorbing" the noise effect into the covariance matrix K. In the later discussion, we will consider the noise as part of the definition of the kernel function for notational convenience.

Now, consider again the noisy training set Ω. We denote the set of a new noisy test points as X_*, which are induced by the stochastic effect in the action parameters. The corresponding targets are denoted $\mathbf{q}_*$, containing a sequence of predictive values with respect to the input in X. With the assumption of a noisy kernel $k(\mathbf{x}_i, \mathbf{x}_j) + \sigma^2 \delta_{ij}$ as the covariance function, the predictive distribution over new targets is thus given by $\mathbf{q}_* \mid \mathbf{f}, \mathbf{X} \sim N\left(\bar{\mathbf{q}}, \sigma[X]\right)$, where

$$\bar{\mathbf{q}} = K(X_*, X)K(X, X)^{-1}\mathbf{q} \tag{5.51}$$

$$\sigma[X] = K(X_*, X_*) - K(X_*, X)K(X, X)^{-1}K(X, X_*) \tag{5.52}$$

Here we note that the derivation to arrive at (5.51) and (5.52) is similar to the weight-space model; however, the interested readers may refer to [14, 107] for more details. With only one test point, (5.51) is reduced to

$$\bar{q}_* = \mathbf{k}(\mathbf{x}_*, X)^{\mathrm{T}} K(X, X)^{-1} \mathbf{q} = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_*, \mathbf{x}_i), \tag{5.53}$$

where in (5.53), $\boldsymbol{\alpha} = K^{-1}\mathbf{q}$ and $\mathbf{k}(\mathbf{x}_*, X)$ is a column vector with component: $\mathbf{k}(\mathbf{x}_*, \mathbf{x}_i), \mathbf{x}_i \in X$. Notice that (5.53) is effectively a kernelized version of (5.10) and is analogous to (5.17) represented by an explicit basis set. Moreover, notice from (5.53) that the mean prediction $\bar{q}_*$ can be expressed as a linear combination of the kernels which effectively associate pair-wise comparisons between the new test point $\mathbf{x}_*$ and the old inputs $X = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_m]^T$ one by one through their inner products implicitly computed by $k(\mathbf{x}_*, \mathbf{x}_i)$. Further, by Mercer's Theorem, any non-degenerative kernel over real inputs (i.e. $\mathbf{x}_i, \mathbf{x}_* \in \mathbb{R}^n$) assumes an expansion of its eigenfunctions:

$$k(\mathbf{x}_*, \mathbf{x}_i) = \sum_{j=1}^{\infty} \lambda_j \phi_j(\mathbf{x}_*) \phi_j(\mathbf{x}_i) \tag{5.54}$$

Equation (5.54) together with (5.53) suggests that using GPR with a non-degenerative kernel is equivalent to the Bayesian linear regression model mentioned in Section 5.2 using infinitely many basis functions. We shall see in Chapter 6 how this generalization property can help us predict the value of an unseen state with stochastic action effects.

5.5 Reproducing Kernel Hilbert Space

The functional form of the mean GP prediction given in (5.53) serves as a basis for representing input patterns in a kernelized form. In Chapter 6, we shall see that the kernelized representation for inputs (e.g. generalized state-action pairs) facilitates the notion of policy generalization within the neighboring or correlated states. In particular, the mean prediction can be viewed as a linear combination of observed signals weighted by their corresponding kernel terms. Writing (5.53) in the form of an operator gives:

$$Q(\cdot) = \sum_{i=1}^{m} \alpha_i k(\cdot, \mathbf{x}_i), \tag{5.55}$$

where $\boldsymbol{\alpha} = K^{-1}\mathbf{q}$. The value prediction for an arbitrary test point $\mathbf{x}$ is thus largely determined by the "relevant evidences" where the associated kernels $k(\mathbf{x},\mathbf{x}_i)$ evaluate to relatively higher values. By Mercer's Theorem, any positive-definite kernel $k(\cdot,\cdot)$ effectively computes the inner product in the kernel-induced feature space: $k(\mathbf{x},\mathbf{x}_i) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}_i) \rangle$, where Φ is a function that maps the input $\mathbf{x}$ to a feature vector with components $\sqrt{\lambda_i}\phi_i(\mathbf{x})$ and $i = 1,2,...,N_H$ (also see (5.54)). In other words, the feature map Φ transforms the input $\mathbf{x}$ to the Hilbert space defined by the weighted inner product:

$$\langle \boldsymbol{\varphi}, \tilde{\boldsymbol{\varphi}} \rangle = \sum_{i=1}^{N_H} \sqrt{\lambda_i}\varphi_i \sqrt{\lambda_i}\tilde{\varphi}_i = \sum_{i=1}^{N_H} \lambda_i \varphi_i \, \tilde{\varphi}_i \tag{5.56}$$

With (5.56), the inner product of two feature vectors thus satisfies:

$$\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}_i) \rangle = \sum_{j=1}^{N_H} \lambda_j \phi_j(\mathbf{x}) \phi_j(\mathbf{x}_i) = k(\mathbf{x},\mathbf{x}_i) \tag{5.57}$$

Again, $N_H \to \infty$ for non-degenerative kernels.

Further, the class of functions in the form of (5.55) allows us to draw a connection between kernelized input patterns (i.e. $\{k(\cdot,\mathbf{x}_i) \mid \mathbf{x}_i \in X\}$) and their contributions to the value prediction through identifying a valid inner product between any two kernelized patterns. To this end, first notice that $k(\cdot,\cdot)$ is symmetric and thus, Φ in (5.57) suggests the mapping: $\Phi : \mathbf{x}_i \to k(\cdot,\mathbf{x}_i)$ such that Φ effectively maps the input domain X to a space of functions $f : X \to \mathbb{R}$ that evaluate any test input pattern $\mathbf{x}$ in terms of $k(\mathbf{x},\mathbf{x}_i)$; i.e. $(\Phi(\mathbf{x}_i))(\mathbf{x}) = (k(\cdot,\mathbf{x}_i))(\mathbf{x}) = k(\mathbf{x},\mathbf{x}_i)$. Thus, taking a linear combination of kernelized input patterns $\{k(\cdot,\mathbf{x}_i) \mid i = 1,...,m\}$ forms a vector space. Now, consider another function of the same form as Q:

$$Q'(\cdot) = \sum_{i=1}^{m'} \alpha_j' k(\cdot,\mathbf{x}_j) \tag{5.58}$$

where $m' \in \mathbb{N}$, $\alpha_j' \in \mathbb{R}$, analogous to the parameters for Q, and $\mathbf{x}_j \in X$. By virtue of the symmetric property inherent in both $\langle \cdot, \cdot \rangle$ and $k(\cdot, \cdot)$, one can then postulate that the inner product of the two functions Q and Q' can be well defined in the following form:

$$\langle Q, Q' \rangle = \sum_{i=1}^{m} \sum_{j=1}^{m'} \alpha_i \alpha_j' k(\mathbf{x}_i, \mathbf{x}_j) \tag{5.59}$$

The next step is to verify that (5.59) does compute a valid inner product. Notice that (5.59) can be expressed explicitly as a linear combination of both Q and Q'; that is,

$$\langle Q, Q' \rangle = \sum_{j=1}^{m'} \alpha_j' \left(\sum_{i=1}^{m} \alpha_i k(\mathbf{x}_j, \mathbf{x}_i) \right) = \sum_{j=1}^{m'} \alpha_j' Q(\mathbf{x}_j) \tag{5.60}$$

Similarly,

$$\langle Q, Q' \rangle = \sum_{i=1}^{m} \alpha_i \left(\sum_{j=1}^{m'} \alpha_j' k(\mathbf{x}_i, \mathbf{x}_j) \right) = \sum_{i=1}^{m} \alpha_i Q'(\mathbf{x}_i) \tag{5.61}$$

Equations (5.60) and (5.61) imply that $\langle \cdot, \cdot \rangle$ is both symmetric and bilinear since $\langle Q, Q' \rangle$ is linear in both Q and Q', and $\langle Q, Q' \rangle = \langle Q', Q \rangle$. The symmetric property also follows from the fact that $k(\cdot, \cdot)$ is symmetric. Further, the positive definiteness of $k(\cdot, \cdot)$ suggests that for any function Q (and Q'), $\langle \cdot, \cdot \rangle$ always evaluates to a nonnegative value; that is,

$$\langle Q, Q \rangle = \|Q\|_H^2 = \sum_{i,j=1}^{m} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \geq 0 \tag{5.62}$$

With the inner product definition in (5.59), one can thus evaluate the inner product between Q and a kernelized pattern $k(\cdot, \mathbf{x})$ in terms of

$$\langle Q, k(\cdot, \mathbf{x}) \rangle = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}, \mathbf{x}_i) = Q(\mathbf{x}) \tag{5.63}$$

In particular, consider the case where there is only one observation in our training set $\Omega : \{(\mathbf{x}_1, q_1)\}$, which corresponds to a single kernelized pattern: $k(\cdot, \mathbf{x}_1)$. Taking the inner product between $k(\cdot, \mathbf{x}_1)$ and the test pattern $k(\cdot, \mathbf{x})$ using the definition in (5.59) gives

$$\langle k(\cdot, \mathbf{x}_1), k(\cdot, \mathbf{x}) \rangle = k(\mathbf{x}, \mathbf{x}_1) = \langle \Phi(\mathbf{x}_1), \Phi(\mathbf{x}) \rangle = \Phi(\mathbf{x}_1)(\mathbf{x}). \tag{5.64}$$

In (5.64), the first equality follows from taking $Q(\cdot)$ to be $k(\cdot, \mathbf{x}_1)$; the second equality again follows from Mercer's Theorem. In view of the property illustrated in (5.63) and (5.64), any positive-definite kernels $k(\cdot, \cdot)$ are also called *reproducing kernels*. The mean GP prediction in the form of (5.55) thus forms a vector space with the kernel satisfying the reproducing property given in (5.63). Such a vector space is called Reproducing Kernel Hilbert Space, RKHS. The formal definition of RKHS is given in Definition 5.4. Additionally, each positive-definite kernel *k* can be shown to correspond to a unique RKHS and conversely, a RKHS also uniquely determines *k* (Theorem 5.5).

Definition 5.4 (**Reproducing Kernel Hilbert Space**) [22, 36] *Let H be a Hilbert space of real functions f defined on an (non-empty) index set X:* $f : X \to \mathbb{R}$. *Then H is called a reproducing kernel Hilbert space endowed with an inner product* $\langle \cdot, \cdot \rangle$ *and the norm* $\|f\|_H = \sqrt{\langle f, f \rangle}$ *if there exists a function* $k : X \times X \to \mathbb{R}$ *with the following properties:*

a) For every $\mathbf{x}$, $k(\mathbf{x}, \mathbf{x}')$ *as a function of* $\mathbf{x}'$ *belongs to H; i.e. k spans H.*

b) k has the following reproducing property:

$$\langle f, k(\mathbf{x}, \cdot) \rangle = f(\mathbf{x}) \text{ for all } f \in H.$$

In particular,

$$\langle k(\mathbf{x}, \cdot), k(\mathbf{x}', \cdot) \rangle = k(\mathbf{x}, \mathbf{x}').$$

Theorem 5.5 (**Moore-Aronszajn Theorem**) [36]. Let X be an (non-empty) index set. Then for every positive definite function $k(\cdot,\cdot)$ on $X \times X$ there exists a unique RKHS, and vice versa.

By virtue of the reproducing property, one can think of the training set Ω given earlier as referencing a set of kernelized input patterns $\{k(\cdot,\mathbf{x}_i) \mid i = 1,...,m\}$ and their individual contributions to the final value prediction are proportional to the corresponding signal strengths $\{q_i\}$. Specifically, the value prediction for a new test point **x** can be achieved by first kernelizing **x** to give $k(\cdot,\mathbf{x})$, followed by taking the inner product with the current value estimate in the form of (5.55):

$$\langle Q, k(\cdot,\mathbf{x})\rangle = \sum_{i=1}^{m} \alpha_i \langle k(\cdot,\mathbf{x}), k(\cdot,\mathbf{x}_i)\rangle = \sum_{i=1}^{m} \alpha_i k(\mathbf{x},\mathbf{x}_i) = Q(\mathbf{x}). \tag{5.65}$$

5.6 Turning RL into a Kernel Machine

The parametric-action model attempts to incorporate a richer expression into the standard MDP formalism through identifying the localized, transient dynamics caused by the process of actions carried out in the state space. In addition, through the introduction of action parameters, it opens up the possibility to form an action-oriented abstraction over the state space as we have seen in the CAQ-learning algorithm (see Section 4.4), which also has a potential to reduce the state space in the decision process. However, in order to take a step further to achieve effective dimension reduction in general, it is not sufficient to address the issue merely through the clustering in the action parameter space. The difficulty lies in finding an efficient yet automatic mechanism for controlling, especially from the perspective of feature selection, the degree to which the action parameters are correlated to the state features in order to formulate effective compression through clustering action parameters. Further, in order to make effective predictions over the parameterized actions, it is imperative to identify the

functional relationship that relates the context (i.e. the state) in which the parametric action was applied to a utility value which, by definition, encodes the fitness of the resulting state-action pair through long-term accumulated rewards.

As we have seen earlier in Section 5.1, the standard linear model can be "kernelized" (e.g. compare (5.26) and (5.27)) in order to express the functional structure with complex, non-linear properties. In principle, any algorithm that can be expressed in the form of inner products can be kernelized. In particular, if an algorithm is formulated in terms of a positive-definite kernel $k(\cdot,\cdot)$ (e.g. kernel ridge regression) then the same algorithm can be constructed by another positive-definite kernel $k'(\cdot,\cdot)$. This again is also known as the kernel trick. Thus, the role of the kernel as pair-wise similarity measure through the inner product of training samples in value prediction can be used to identify the functional structure of interest within a generalized state-action pair involving actions expressed in vectorial forms. This allows us in some sense to covert the parametric action model into an MDP-compatible form (or POMDP-compatible by extension) on a higher level through the concept of the kernel that reduces the parametric action in a multivariate representation into a concrete set of actions. Certainly, additional work is required to precisely describe such conversion as we shall see in Chapter 6. Nonetheless the objective is for the new model to still enjoy the property of parametric actions from the lower level while the standard MDP formulation can be used to bear upon the decision process that takes place on the level of kernelized action representation.

Other examples of kernel machines include: kernel K-means [108, 109], kernel PCA [108], GPR [14], and spectral clustering [24], among many others. Kernel k-means algorithms are essentially formulated by kernelizing the cluster objective function (e.g. the sum-of-square error function of data points with respect to the centroid). In this manner, the cluster assumption can be singled out as another layer of the learning process on top of the underlying clustering algorithm. This could help in systematically approaching the best data similarity assumption by which a complex data distribution can be well-explained by clusters. In fact, weighted kernel k-

means, formulated by assigning weights to each data point in the objective function, can be shown to have its equivalency with spectral clustering [109] since the derivation towards their solutions both trickle down to the same type of trace maximization problem, which we will briefly cover in Chapter 7. Similarly, kernel PCA is formulated by kernelizing the data covariance matrix used to identify the intrinsic dimension of the data (also known as the *principle subspace* of the data [107]) along which the data distribution has the largest variance. Kernel PCA can also be shown to have its equivalency with kernel k-means in approaching their solutions [108]. With these interrelated kernel machines in mind, we will further investigate GPR as an adaptive value-function predictor and spectral clustering as the mechanism in formulating action abstraction later on in Chapter 6 and Chapter 7, respectively.

CHAPTER 6

REINFORCEMENT FIELD

The construct of parametric actions introduced in Chapter 4 enables the agent to express more complex actions that may not be easily represented using a concrete enumeration of available action choices. This representation facilitates the grouping of similar actions in terms of clustering with respect to their reinforcement feedbacks. The forming of a cluster-based conceptual model reduces the number of action choices required to learn a reasonable control policy. Moreover, through the process of action cluster formation, the state space is correspondingly partitioned into a set of coherent regions provided that the state features are dependent on the action parameters and the functional relation for such dependency is known. CAQ-learning was introduced as an example of embedding such action abstraction into an existing RL algorithm. However, a few strong assumptions were made in order to simplify such action-model embedding. For instance, the correlation between action parameters and state features was assumed to be known a priori. Also, the action similarity was defined only with respect to action parameters by assuming that the structure of the action cluster is approximately identical across the state space, giving rise to a global action model. Further, the reward function was effectively used as a heuristic for implementing the cluster assumption for grouping actions. In this chapter, our first goal toward a more generalized learning architecture is to drop the assumptions above by making use of the property of kernel functions in a regression model. Similar to the earlier chapters, the architecture schematic as a roadmap towards the generalized RL framework is given in Figure 6.1, in which the logical components to be covered in this chapter are colored. In retrospect, it may be helpful to compare Figure 6.1 with the earlier schematics in the previous chapter beginnings.

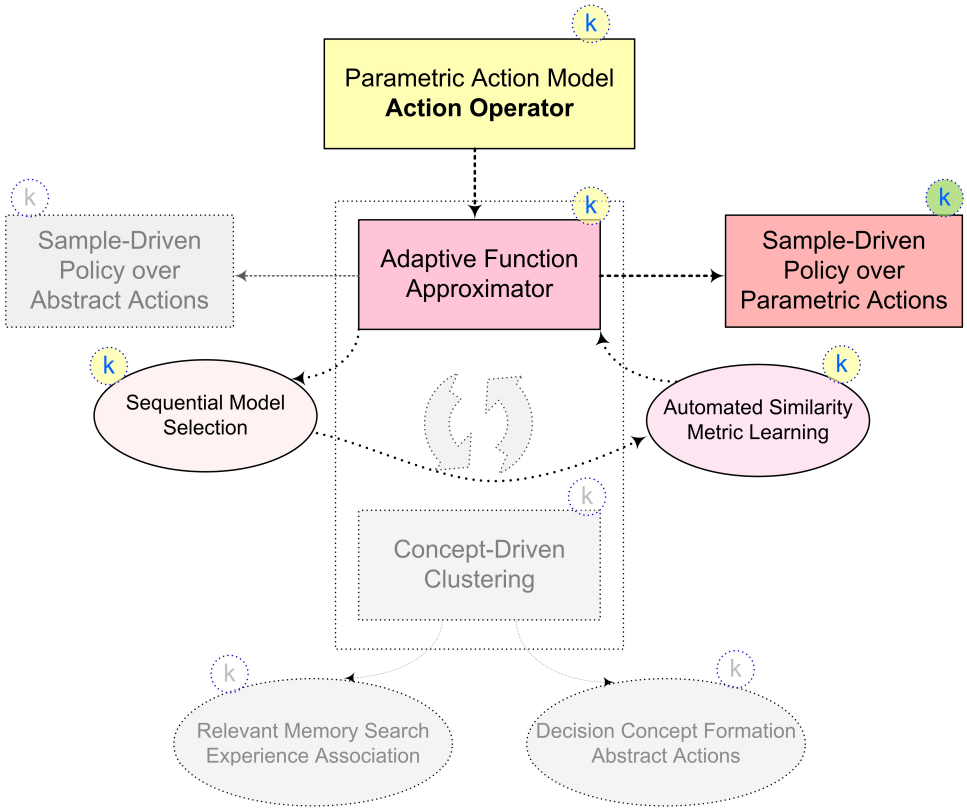


Figure 6.1 The role of the value predictor and its peripheral components in the generalized RL framework. The action formulation to be introduced in this chapter extends from the earlier parametric action model in a manner that integrates seamlessly with the value predictor. Using GPR to represent the value predictor facilitates the notion of sequential model selection, which in turn accomplishes the learning of the similarity measure for the control decisions made across different contexts.

6.1 The General Approach: A Roadmap

Complex domains such as the task-assignment problem in the Grid often require an adaptive policy that is responsive to the constantly-changing environment. Recall from Chapter 4 that the standard reinforcement learning framework establishes a unique way of deriving such control policies over the state space representation in which the policy unfolds as a sequence of actions, guiding the agent towards a trajectory with a maximized aggregate reward. In particular, a potential function maps input states or state-action pairs to their corresponding values (e.g. the action-value function in (4.6)), which serve as a fundamental basis for

evaluating a control policy. In order to allow for a generalization into a large, continuous state space, function approximation techniques [2, 4, 5] are often employed to express such potential functions in a more compact form. In Chapter 5, we have explored basic regression methods that could serve as the function approximator for the purpose of policy learning. Nonetheless, with the representation in standard RL algorithms, the complexity for policy learning largely depends on the dimensionality of the state space which in turn governs the required size of the basis set and how effectively potential function values (or utilities) can be evaluated. Real-world control tasks, however, often involve a large state space. An extreme case can be observed when using multiagent approach to address decentralized MDP problems such as applying the cooperative multiagent MDP framework [30] for the task assignment domain mentioned earlier. In such framework, since user tasks and compute resources are represented by their individual state and action spaces, learning a task assignment policy requires forming a join state space as the Cartesian product of the states of all tasks and all candidate machines. The computational cost of maintaining a global system state as such can be exceptionally high if not intractable for a large Grid network [21] even without dealing with the extra factor of a varying set of candidate machines induced by the metagrid introduced in Chapter 2.

One of the core issues in the standard RL framework lies in the fact that the control policy depends on a concrete set of actions, modeling allowable behaviors of the agent, while the overall behaviors themselves are often dynamic processes. Furthermore, once an action is executed, the outcomes may not be as consistent as expected due to the stochastic nature of the environment or of the actuators themselves. Indeed, enabling a richer expression for actions was the primary incentive for introducing the parametric-action model in Chapter 4. However, we have not addressed specifically how these action parameters can be modeled in a manner that can be integrated effectively into state transitions and the policy search. To do so, we will begin by expressing each action parameter as a constrained random process associated with a task-dependent probability distribution. Next, we will further generalize parametric actions

collectively into an *action operator, a* construct that explicitly encodes the dynamic process of state transitions. That is, a state transition induced by an action is specified operationally in terms of the action operator taking on an input state and subsequently producing the next state. The *action operator* is also linked to the potential function through action parameters, enabling the result to be predictable whenever acting on states. Further, with actions being parameterized in a manner that exhibits properties of a continuous function, we will establish the notion of the *reinforcement field* as a policy generalization mechanism. The property of the field comes from using the Gaussian process approach as a kernel machine in which the kernel represents an adaptive correlation hypothesis over samples of experiences while operating on the vector field in Hilbert space. The potential function in the reinforcement field is driven by an underlying control learner using a selected temporal-difference learning algorithm (see Chapter 4) for value predictions. In particular, the utility estimate from the control learner is taken to represent a feedback signal, defined later as a fitness value, in response to the action operator acting on a state. The exact behavior of the action operator depends on the action parameters with their values stochastically determined by the related random processes. Interested readers could also refer to other types of GP-based RL methods in [10, 11], where GP is often used to represent the environment dynamics including the state transition, observation and reward functions, etc.

6.2 Action Dynamics

In order to express the notion that actions are not merely immutable choices with behavioral details predefined, the first step in generalizing the standard RL framework is to introduce parameters to actions. In Chapter 4, we have seen the action parameterization from the perspective of modeling the “side effect” introduced as the action is performed in a state. In particular, a navigation domain was presented where the side effect corresponds to the removal of objects upon visiting a particular location. This time, however, we shall motivate the

parametric-action model from the perspective of expressing *action dynamics*. The rationale follows from the fact that in reality, executing an action involves a variational procedure over the configuration of the active state; that is, the state feature values change in response to the continuous process during which an action is applied. The variation could come from the unpredictability of the actions when performed, such as the error introduced by misaligned actuators. Affordance or feasibility of an action can also cause variational impact over a state. For instance, the result of the task assignment to a candidate machine is subject to the time-varying machine profile, which may at one time exhibit a rich computational resource but at other times become completely drained in resource capacity for the incoming task. The variational process by definition certainly also covers the case of localized effects such as the consumption or removal of objects as mentioned in the earlier chapter.

The localized effect induced by the action can be expressed in terms of the variation within a subset of state features such that a group of states may share the same partial variations. For instance, STRIPS operators express such action effects through propositional logic and first-order predicate calculus. The action representation using Dynamic Bayesian Networks (DBN) from the factored-MDP framework [96, 97] extends this concept further into stochastic actions. Specifically, an action in DBN triggers a state transition in terms of the probability conditioned only on a subset of state variables relevant to the predecessor state. However, in order to further express the fluid aspect of the action with a dynamic description as it acts upon a state, we shall see the parametric-action model offers certain advantages that go beyond the decision-theoretic perspective that perceives an action as a logical choice.

Parameterizing the action effectively introduces extra degrees of freedom necessary to express action dynamics. Again we will refer to a primitive action in terms of the notation: $a \in A$ in contrast to an explicitly parameterized action $a(\mathbf{x}_a) \in A,$ where $\mathbf{x}_a$ is the action vector in the parametric action space A^+. In this manner, each primitive action is represented by relevant features that encode specifically what local variations can occur when it is chosen to act on a

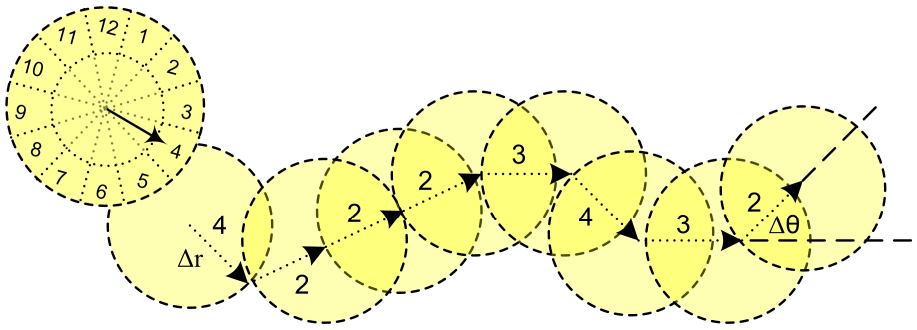


Figure 6.2 Primitive actions parameterized by a radius and an angle random variable: both of the action parameters manifest as a random process when their associated action choices are applied in a sequence of states.

state. Figure 6.2 reintroduces a 2D navigation domain in which the actions of the agent are parameterized by an angle and a radius increment per move relative to the current position; i.e. $\mathbf{x}_a = (\Delta r, \Delta\theta)$.

However, we cannot apply, without an extension, the update rule from temporal-difference learning introduced earlier (e.g. (4.10) for SARSA) under the extended state space induced by the parametric action. Expanding upon the standard RL formulation, three special constructs are defined here that help in realizing the desired properties using the parametric-action model: i) augmented state space (Definition 6.1) ii) potential function over augmented state space (Definition 6.2) and iii) experience particles (Definition 6.3).

Definition 6.1 (**Augmented State Space**) *The augmented state space* S^+ *is a tensor product of the state space* S *and the action parameter space* A^+: $S^+ = S \otimes A^+$.

Definition 6.2 (**Fitness Function**) *A fitness function is a potential function* $Q^+ : S^+ \to \mathbb{R}$ *that maps an augmented state* $s^+ \in S^+$ *to a fitness value* $Q^+(s^+)$.

Definition 6.3 (**Experience Particle**) *An experience particle is an abstraction of a functional datum consisting of the tuple* $(s^+, Q^+(s^+))$ *representing an augmented state* s^+ *and its corresponding fitness value* $Q^+(s^+)$.

Note that A^+ in Definition 6.1 refers to the parameter space consisting of action vectors; any action vector when paired with an active state (where the action is feasible) resolves to an augmented state in terms of a tensor product element. Although any arbitrary probability distribution can be potentially used to model the stochastic effect of the action, in this research, we will use a *yield function* [50] to model each coordinate of the action vector. Specifically, a *yield function* assumes the following form:

$$Y(x_i) = P((x_i + w_i) \in \Gamma), \tag{6.1}$$

where x_i corresponds to the value of the *ith* feature dimension of the action vector (i.e. $(\mathbf{x}_a)_i$), w_i represents a random variation along the *ith* dimension and Γ is the set of acceptable values (i.e. support). In addition, a probability threshold $\eta \in [0,1]$ is used as a control parameter such that $(\mathbf{x}_a)_i$ takes on the value of x_i for which $Y(x_i) \geq \eta$; i.e. $(\mathbf{x}_a)_i = x_i \mid Y(x_i) \geq \eta$. Note that the threshold η effectively determines the *η-yield region* in which $x_i + w_i$ will fall within Γ with a probability at least *η*. In particular, the target value x_i models the expected local variation by which the related state feature will shift upon a transition to the next state. This corresponds to a "normal behavior" of the action along this coordinate. The uncertainty in action performance is modeled by w_i which is chosen to follow a task-dependent probability distribution. For instance, a Gaussian distribution is often a reasonable choice. An action parameter value outside the boundary of Γ is taken to be either infeasible or empirically improbable by assuming η to be close to 1. In Figure 6.2, the left-most circle indicates 12 possible orientations of movements in which each pie-shaped boundary in between two concentric circles represents a feasible range of the motion in terms of the action parameters: $(\Delta r, \Delta\theta)$. For instance, action 3 represents a motion moving along the direction to the east but with a tolerance of any errors within ± 15 degrees. Similarly, the radius is allowed to fall within the inner and outer circles. Note that the uncertainty in action parameters can alternatively be interpreted as the uncertainty inherent in

the observations of states, depending on the choice of approximating an MDP or a Partially Observable MDP (POMDP). Although this dissertation will focus on the MDP approximation, a POMDP analogy will be provided in Section 6.7.3 to further point out the implication of stochastic action effects with more details.

By introducing stochastic action parameters, complex actions can be more easily modeled and thereby incorporated into the policy search. In particular, the action performance is allowed to vary over time reflecting the true scenario in real-world applications. For instance, a robotic arm may not always follow the expected trajectory induced by the forward kinematics to reach a particular spatial region due to small errors in the prescribed axis- or angle-related parameters specifying joint links and their connections. Similarly, a robot navigating in an open area may not always take a step with a fixed length or resolve its orientations without unexpected tilts. Thus, the desired control policy needs to exhibit some capability in fault tolerance to be practically applied in complex, real-world applications.

6.3 Representing Evidences as Functional Data

In the new framework, every time the agent executes a parametric action in a given state, an *experience particle* (Definition 6.3) is created, encapsulating the corresponding augmented state along with a "measure" of its potential value, addressed specifically as the *fitness value* (Definition 6.2). As an illustration, suppose that $\mathbf{x}_s$ and $\mathbf{x}_a$ represent a feature vector for a state $s \in S$ and a primitive action $a \in A$ respectively, then the corresponding augmented state space S^+ can often be considered as the joint vector space spanned by merging the coordinates of the element of the tensor product: $(\mathbf{x}_s, \mathbf{x}_a) \in S \otimes A^+$ to give $\mathbf{x} = \left(x_1^s, ..., x_n^s, x_1^a, ..., x_m^a\right)$, assuming that $\mathbf{x}_s$ has a dimension n and $\mathbf{x}_a$ has a dimension m. The fitness function Q^+ evaluated over an augmented state is given by:

$$Q^+(s^+) = Q^+(\mathbf{x}) = Q^+\left(\left(x_1^s,...,x_n^s,x_1^a,...,x_m^a\right)\right) \tag{6.2}$$

Thus, if the agent traverses the state space with *m* steps, a sequence of training samples will be generated and retained in memory: $\Omega : \{(\mathbf{x}_1, q_1),...,(\mathbf{x}_m, q_m)\} \subseteq X \times Q$, where X and Q, respectively, denote the set of augmented states and their corresponding observed fitness values. It is helpful for the moment to consider these training samples in Ω as a functional data set [39] referenced by the experience particles distributed over the state space. The mechanism for estimating their values are deferred to Section 6.5. Each experience particle effectively defines a control policy that generalizes into the neighboring (augmented) state space through properties of the kernel to be discussed shortly. The similarity of any two particles (or equivalently, two referenced augmented states) takes into account both the state vector $\mathbf{x}_s$ and action vector $\mathbf{x}_a$. This formulation is made possible by allowing the action to take on continuous variables along with the use of a kernel function as a correlation hypothesis associating one augmented state to another through their inner products in the kernel-induced feature space [22]. With the above constructs defined, the next step toward establishing the reinforcement field is to represent the fitness function in a manner that integrates with parametric actions and, in the meantime, serves as a "critic" for the policy embedded in experience particles. In this research, we represent the fitness value function through a progressively-updated Gaussian process.

6.4 Value Predictions using GPR

Consider the training set $\Omega : \{(\mathbf{x}_1, q_1),...,(\mathbf{x}_m, q_m)\}$ given earlier. The goal is to identify a function $Q^+(\mathbf{x})$ (Definition 6.2) fitting these samples with a tradeoff between quality of the prediction (i.e. data fit) and smoothness assumptions of the function (i.e. model complexity) [14]. Value predictions using GPR first assume a normal prior distribution over functions, and subsequently reject those functions not consistent with the observations (i.e. experience

particles). Suppose that we wish to reconstruct the target function Q^+ from the noisy observations of fitness values, i.e. $q_i = Q^+(\mathbf{x}_i) + \varepsilon_i$ where ε_i is assumed to be i.i.d. Gaussian noise such that any set $\{q_i\}$ follows a Gaussian distribution with a covariance function $k(\cdot,\cdot)$. The prior distribution over the observed values $\{q_i\}$ is thus given by: $\mathbf{q} \mid \mathbf{x} \sim N(0,K)$, where $K_{ij} = k(x_i, x_j)$ and the fitness value observations are assumed for simplicity to be centered around a zero mean. In this research, we will explore two different kernels – an SE kernel in (6.3) and a product kernel in (6.4). Equation (6.3) is a noisy squared exponential kernel (SE) that combines all the features and action parameters in the exponential term; i.e.

$$k(\mathbf{x},\mathbf{x}') = \theta_0 exp(-\frac{1}{2}(\mathbf{x}-\mathbf{x}')^{\mathrm{T}} D^{-1}(\mathbf{x}-\mathbf{x}')) + \theta_1 \sigma^2_{noise} \tag{6.3}$$

In (6.3), the diagonal elements in D, signal amplitude θ_0 and the noise magnitude θ_1 correspond to the hyperparamters of the kernel. Alternatively, the state features and action parameters can be represented via two separate kernels so as to distinguish them as two different objects; this gives:

$$k(\mathbf{x},\mathbf{x}') = \theta_0 k_s(\mathbf{x}_s,\mathbf{x}'_s) k_a(\mathbf{x}_a,\mathbf{x}'_a) + \theta_1 \sigma^2_{noise}, \tag{6.4}$$

where k_s and k_a represent the state kernel and action kernel, respectively, both assumed to be an SE kernel in this research. After observing a series of experience particles, the predictive distribution for the value of a novel augmented state $\mathbf{x}_*$ is given by: $q_* \mid X, \mathbf{q}, \mathbf{x}_* \sim N(\bar{q}_*, \sigma)$ where,

$$\bar{Q}^+(\mathbf{x}_*) = \bar{q}_* = \mathbf{k}(\mathbf{x}_*, X)^{\mathrm{T}} K(X,X)^{-1}\mathbf{q} \tag{6.5}$$

$$\sigma = k(\mathbf{x}_*,\mathbf{x}_*) - \mathbf{k}(\mathbf{x}_*, X)^T K^{-1}\mathbf{k}(X,\mathbf{x}_*) \tag{6.6}$$

In (6.5) and (6.6), the set of augmented state vectors as training samples are stacked in X as column vectors for convenience; thus, $\mathbf{k}(\mathbf{x}_*, X)$ represents a column vector with each

entry evaluated by $\mathbf{k}(\mathbf{x}_*, \mathbf{x}_i)$. In particular, the mean prediction following (6.5) is used for evaluating the fitness value Q^+ for a given augmented state.

The choice of using GPR as a representation for the fitness function boils down to the consideration of a few desired properties. First, the GP-based fitness function assimilates both the state features and action parameters through the kernel as a covariance function such that the similarity in the prediction also reflects the degree of similarity of the policy. Secondly, since the value prediction is driven by the samples (i.e. experience particles) having certain correlation to one another (measured by the kernel), the policy can theoretically be deduced everywhere in the state space including those regions not yet being explored. That is, as long as there exist sufficiently correlated particles in the proximity, the kernel as a correlation hypothesis for particles will continue to apply without requiring the agent to explicitly explore the unknown region. This is in contrast to the regression model with a fixed set of basis functions introduced in Chapter 5 (see Section 5.2.3 for more details). We shall see that this property along with the action operator (to be defined in Section 6.6) have benefits that build upon each other.

Further, the kernel in GPR adapts to the ongoing variations in the environment dynamics by the procedure of Automatic Relevance Determination (ARD) [14] that tunes the kernel hyperparameters using conjugate gradient optimization. In particular, the ARD procedure performs an automatic model selection by maximizing the marginal likelihood of the data:

$$\log p(\mathbf{q} \mid X) = -\frac{1}{2}\mathbf{q}^T K^{-1}\mathbf{q} - \frac{1}{2}\log|K| - \frac{n}{2}\log 2\pi, \tag{6.7}$$

where entries in K are evaluated through a given kernel (e.g. (6.3), (6.4), etc). The only term in (6.7) that involves the observed fitness values represents the data fit during policy learning: $-\mathbf{q}^T K^{-1}\mathbf{q}/2$ while the term $-\log|K|/2$ represents the complexity of the fitness-value model $Q^+(\cdot)$. The ARD process corresponds to maximizing (6.4) by incorporating a separate

hyperparameter for each feature of the augmented state (i.e. for each coordinate of the state and the parameterized action). This helps in determining the degree of relevance that each feature contributes to the final prediction. To determine the optimal hyperparameter, first express explicitly the marginal likelihood function (6.7) conditioned on the hyperparameter as $\log p(\mathbf{q} \mid X, \boldsymbol{\theta})$, followed by taking partial derivatives of (6.7) with respect to all hyperparameters to give:

$$\frac{\partial}{\partial \theta_i} \log p(\mathbf{q} \mid X, \boldsymbol{\theta}) = \frac{1}{2} \mathbf{q}^T K^{-1} \frac{\partial K}{\partial \theta_i} K^{-1} \mathbf{q} - \frac{1}{2} tr\left(K^{-1} \frac{\partial K}{\partial \theta_i} \right) \tag{6.8}$$

By (6.8), the complexity of using ARD to determine the degree of relevance for each augmented state feature toward the Q^+ value prediction is largely dominated by the matrix inversion of K. In Section 6.7, we shall see an algorithm that uses the property of temporal differences to preserve only the essential experience particles necessary for the policy inference, thereby reducing the dimension of the covariance matrix for computational advantage.

6.5 Representing the Policy

Since the objective of the fitness function Q^+ is to act as a critic for experience particles, the policy is expected to take on the form of a functional depending on Q^+; i.e. $\pi[Q^+]$. Using a softmax function, the policy over a pair of state and parametric action (referenced by a particle) can be represented by:

$$\pi\left(s, a^{(i)}\right) = \frac{\exp\left[Q^+\left(s, a^{(i)}\right) / \tau\right]}{\sum_j \exp\left[Q^+(s, a^{(j)}) / \tau\right]}$$

$$= \frac{\exp\left[Q^+\left((\mathbf{x}_s, \mathbf{x}_a^{(i)}\right) / \tau\right]}{\sum_j \exp\left[Q^+\left((\mathbf{x}_s, \mathbf{x}_a^{(j)}\right) / \tau\right]} \tag{6.9}$$

Although $(\mathbf{x}_s, \mathbf{x}_a^{(j)})$ in (6.9) is in general an element of the tensor product $S \otimes A^+$, the GP-based fitness value predictor in many cases transforms this tensor element into a (weighted) joint vector representing an augmented state. This can be seen, for instance, by comparing (6.2) with the functional form in (6.3). Consequently, $Q^+\left((\mathbf{x}_s, \mathbf{x}_a)\right)$ is simplified to $Q^+(\mathbf{x})$ for notational convenience where $\mathbf{x}$ combines the coordinates of the state and the parametric action. To obtain values for Q^+ for each augmented state, GPR works in parallel with an underlying control learner such as SARSA (see Chapter 4) that "perceives" parametric actions (i.e. $a(\mathbf{x}_a) \in A$) as individual choices (i.e. primitive actions $a \in A$) without knowing the underlying parameters that govern the action dynamics. That is, from the perspective of the control learner, executing a parametric action leads to an update of a utility estimate as usual. In the case of SARSA, this effectively corresponds to the update rule in (4.10). The utility estimate for a given state-action pair (s,a) subsequently propagates to GPR to serve as a training signal for the associated augmented state. In this manner, the complexity for the fitness value prediction need not be constrained by the size of the state space. For instance, the SARSA agent can be chosen to operate on a set of discretized state partitions $S^{(k)} \subseteq S$ so that each fitness value for an augmented state is effectively also an approximate utility value for the action applied in the underlying state partition. Each state partition $S^{(k)}$ maintains a reference to several particles, each of which has an assigned utility estimate, reflecting the subtle differences in the exact values of the augmented states. Specifically, this amounts to the partition $S^{(k)}$ referencing the set $\left\{(\mathbf{x}_s, \mathbf{x}_a)_j^{(k)}\right\}$ with $(\mathbf{x}_s)_j^{(k)}$ falling within the boundary of $S^{(k)}$, where the subscript index distinguishes different particles. Doing so implies that there is a trade-off between the accuracy in the utility estimate and the granularity of the underlying state partitions (i.e. the particle density).

6.6 Reinforcement Field

The reinforcement field combines both the insights from various field theories in physics (e.g. [13]) as well as properties of the kernel function. In the most abstract sense, a field acts on an object (e.g. a particle) in a manner governed by the law of the field. For instance, a charged particle moving in a magnetic field is subject to a Lorentz force that induces a circular motion on the particle. The magnetic field implicitly "prescribes" a particular state sequence for the particle to follow during the course of its travel. The circular motion can be justified by the principle of least action [12] in which an action, for the case of one particle, is defined in terms of the Lagrangian L of the form: $\int_{t_1}^{t_2} L(q(t), q'(t))dt$, where q refers to a generalized coordinate of a physical quantity of interest such as the position of the particle. The action in this formulation essentially represents a global constraint for the particle trajectory. Incidentally, the term action overlaps with that in the RL terminology. Minimizing the action with respect to $q(t)$ (and $q'(t)$) using calculus of variation leads to the Euler-Lagrange equation that effectively represents the local dynamics of the particle, i.e., the motion equations for the particle. Thus, we see that in a physical system, there exists a dynamic process that drives state transitions systematically towards a certain objective (e.g. the path of least time for a light beam) in a manner consistent with the law of the field. By connecting this analogy to the RL framework, the objective of maximizing the accumulated reward corresponds to a global constraint while the local dynamics can be expressed by the parametric actions. An action in this sense assumes the property of an operator often in a functional form (e.g. the integral operator with the Lagrangian $\int L(\cdot)dt$) over a set of related parameters (e.g. position and velocity). For the reason stated above, this research seeks to generalize the parametric action further into an operator-like construct with tunable parameters characterizing the local dynamics of state transitions, and simultaneously links it to the fitness function.

6.6.1 Action Operator

An operator [104] maps the input vector to another vector according to the rule of the operation. A rotation matrix, for instance, is an operator that takes an input vector and rotates it by an angle. The linear operators $T(\cdot)$ such as rotation matrices, integration against kernels given in (5.44), etc, satisfy linearity that can be summarized by the following equalities:

$$T(\alpha\mathbf{x}+\beta\mathbf{x}')=T(\alpha\mathbf{x})+T(\beta\mathbf{x}')=\alpha T(\mathbf{x})+\beta T(\mathbf{x}').$$

In other words, the act of scaling (or summing), before or after applying the operator to the input vectors, will produce the same result. Action operators to be discussed in this section, however, are not limited to linear operators but nonetheless define a specific operation over any input vector representing the current state and subsequently produce its output as the successor state. Readers who are familiar with postulates in quantum mechanics may draw an analogy to the notion of using Hermitian operators [117] to express observables such as position, momentum, etc. Since any external measurement imposed on a particular state of the system (e.g. an electron polarized along an axis) will shift the configuration of the state according to the Heisenberg uncertainty principle, Hermitian operators can thus be used to express the state transition induced by observing the observable. The notion of the operator (or operations in general) provides a systematic framework in specifying how external forces (e.g. actions, the act of measurements, etc) shift the state of the system. Yet in the context of RL, we are also interested in the value associated with the operator, which should reflect the long term payoff after applying the action to a particular state. As a consequence, the action operator not only depends on the state and the parametric action of choice but is also required to integrate with the potential value predictor in a manner that allows for value prediction to occur. Additionally, the action operator does not assume a fixed form until the random effects within action parameters are resolved.

For the sake of discussion, consider three snapshots in a 2D navigation domain in Figure 6.3 where the motion of the agent is again parameterized by the radius and angle

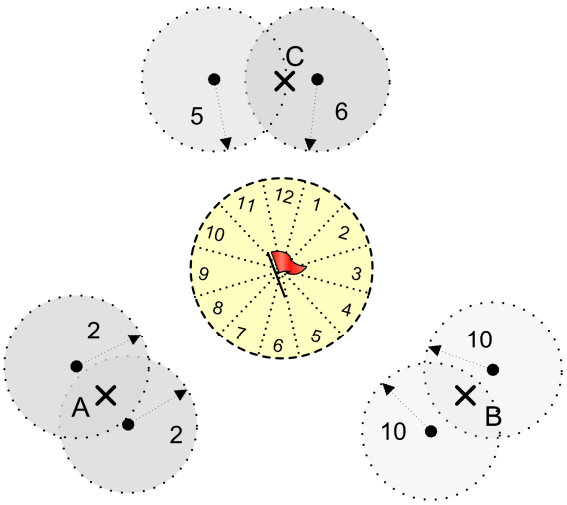


Figure 6.3 Policy inference with experience particles: the control policy in unexplored state space area can be inferred from the neighboring particles referencing known localized policies. Both A and B are within the scope of relatively similar decision contexts where action 2 and 7 are optimal. C is being influenced more strongly by 5, the state on the right.

increment, i.e. $(\Delta r, \Delta\theta)$. Both Δr and $\Delta\theta$ are continuous random variables modeled by yield functions discussed earlier. Given this, one can alternatively represent a particular state transition in terms of the matrix multiplication:

$$\begin{bmatrix} 1+\dfrac{\Delta x}{x} & 0 \\ 0 & 1+\dfrac{\Delta y}{y} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x+\Delta x \\ y+\Delta y \end{bmatrix} \tag{6.10}$$

where $\Delta x = \Delta r \cos(\Delta\theta)$ and $\Delta y = \Delta r \sin(\Delta\theta)$. That is, an action is represented as an operator (e.g. matrix) that takes an input state (x, y) and produces the next state $(x+\Delta x, y+\Delta y)$. Note that the action operator effectively generalizes different primitive actions for the control learner by assuming different parameters within certain bounds. With the action operator, the objective of learning in standard RL is altered slightly to the following statement: *determine the action operator that acts on each state in a manner that leads to a trajectory for a maximum accumulated reward*. Effectively, the action operator serves as a local constraint each state

transition must follow while the fitness function assesses the result of the action operator acting on a state while assuming particular parameters.

6.6.2 Policy Inference in the Field

Recall from (6.5) that the fitness value function is represented through a linear combination of the kernels centered at different augmented states weighted by the observed fitness values. Without explicitly specifying the test augmented state, (6.5) can be rewritten in the following form:

$$\bar{Q}^{+}(\cdot) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \cdot) \tag{6.11}$$

where $\boldsymbol{\alpha} = K^{-1}\mathbf{q}$. Since each $k(\mathbf{x}_i, \cdot)$ in (6.11) effectively maps an input pattern $\mathbf{x}_i$ (an augmented state) into a function, taking a linear combination of these kernels forms a vector space. Consequently, $\bar{Q}^{+}(\cdot)$ in (6.11) can be treated as a vector in a vector field (of functions) with *k* as the *representer of evaluation*. Specifically, $\bar{Q}^{+}(\cdot)$ is a vector in Reproducing Kernel Hilbert Space (RKHS) [1, 22, 23] with the reproducing property: $Q^{+}(\mathbf{x}) = \left\langle Q^{+}(\cdot), k(\cdot, \mathbf{x}) \right\rangle$ where $\left\langle \cdot, \cdot \right\rangle$ denotes an inner product evaluated in the unique RKHS associated with the kernel $k(\cdot, \cdot)$ (also see Section 5.5). By virtue of this property, the known experience particles effectively establish a field through which a novel particle obtains its fitness value by comparing against all the other particles. Each comparison evaluates a particular correlation degree to a known particle. The closer the test point (i.e. the new augmented state in the query) is to a known particle (or more specifically, the augmented state referenced by the particle), the closer their fitness values are. Consequently, the final prediction is governed by those particles in the proximity of the test point. In addition, by Mercer's Theorem [14, 22], any positive-definite kernel assumes an expansion of its eigenfunctions ϕ; that is,

$$k(\mathbf{x},\mathbf{x}_*)=\sum_{i=1}^{N}\lambda_i\phi_i(\mathbf{x})\phi_i^*(\mathbf{x}_*) \tag{6.12}$$

where N approaches infinity if the kernel is non-degenerative such as the SE kernel in (6.3). Note that for the real augmented state vector $\mathbf{x}_* \in \mathbb{R}^n$, $\phi_i^*(\mathbf{x}_*)$ in (6.12) is identical to $\phi_i(\mathbf{x}_*)$. Equation (6.5) together with (6.12) entails that using GPR with a non-degenerative kernel is equivalent to a Bayesian linear regression model using infinitely many basis functions (see also Section 5.3 and 5.4). This effectively implies that the fitness value prediction for a new particle will be bounded by a relatively small variance (see (6.6)) so long as certain experience particles exist in the neighborhood such that their associated augmented states are sufficiently correlated. We are now ready for the definition of a reinforcement field.

Definition 6.4 (**Reinforcement Field**) *A reinforcement field is a vector field in Hilbert space established by one or more kernels through their linear combination as a representation for the fitness function, where each of the kernels centers around a particular augmented state vector.*

As suggested by (6.10) an action operator is a function of both the state features and action parameters. As a result, the exact form of the action operator is not determined until all the parameters in the operator are fixed. Once the state of interest is determined along with the choice of the (primitive) action $a \in A$, the action operator acts on the state such that it resolves the stochastic effects, modeled by the action parameters, into a fixed action vector. This in turn leads to the next state that falls within the range stochastically governed by the action parameters. A fixed action operator is always associated with a fitness value given by (6.5). By this token, an action operator can be applied anywhere in the state space with an associated fitness value evaluated from the knowledge of the environment gathered so far in terms of the experience particles held in memory (i.e. the training set Ω). Further, if the control policy is

defined through a fitness value prediction, such as the softmax policy in (6.9), the action operator also represents an abstraction of a policy. This is due to the fact that the action operator encodes both a particular state transition and its fitness value once the state and action vector are known. Thus, by fixing a state, all (parameterized) action choices can be compared in a similar way through their corresponding fitness value estimates. The result of the fitness value prediction is governed by the property of the reinforcement field, which is established by all the known experience particles distributed over the state space. In this way the reinforcement field connects with the action operator, which, in turn, determines the continuous process of the policy inference. A high-level policy inference in a reinforcement field is illustrated in Figure 6.3. The center circle represents the goal area. Also marked in the center circle are 12 possible motions, each with an allowable range for action variations. The outer three pairs of shaded circles enclose the experience particles (in dots) holding control policies suggested by the associated augmented states. The new query points are marked by crosses. As long as the query point (e.g. an unexplored state) is sufficiently close to the states of the neighboring particles (i.e. within the scope of the shaded circles), the resolved action is expected to be similar (in terms of action parameters) to those in the particles. The exact inference is given in the next section.

To sum up, a reinforcement field shares the following properties analogous to other physical fields: i) a source of origin ii) a spatial intensity and iii) a governing law that acts on particles in the field. The field source originates from experience particles, each of which embeds a control policy that generalizes into the neighboring states through the kernel. The intensity of an experience particle is represented by the fitness value evaluated from the combined effect of all the other particles in proportion to their degree of correlation and their intensities (i.e. their fitness values). Upon any state transition, a new particle is created, encapsulating the value of the state vector, action vector, and the mapping to a fitness value. The new particle joins the rest of the population to form an updated reinforcement field that

subsequently determines the fitness value of yet another new particle. The collection of experience particles corresponds to the functional data set given earlier, i.e. $\Omega : \{(\mathbf{x}_1, q_1), ..., (\mathbf{x}_m, q_m)\}$, in which each element in Ω represents a particle in terms of its (augmented) state and signal strength. For convenience of exposition, denote the action operator explicitly by $A(\cdot)$ and the new augmented state by $\mathbf{x}$ for which the (fitness) value is to be determined. The action operator $A(\cdot)$ can be interpreted as a generalized function (i.e. distribution) defined in terms of a pairing with the test function $k(\cdot, \mathbf{x})$ to produce a real-valued output. In particular, the inner product $\langle Q^+(\cdot), k(\cdot, \mathbf{x}) \rangle$ can be used to evaluate the value of $\langle A(\cdot), k(\cdot, \mathbf{x}) \rangle$ as the distribution of $A(\cdot)$. In this manner, both the state transition and the associated value estimate are simultaneously determined as the operator acts on the state. The logical connection from the action operator to its associated value prediction can be summarized as follows:

1) At a given state $s \in S$, the agent chooses a (parametric) action $a \in A$ according to the current policy $\pi[Q^+]$ (see (6.9)).
2) The action operator resolves the random effect in action parameters through a sampling process such that the (stochastic) action is reduced to a fixed action vector $\mathbf{x}_a$.
3) The action vector resolved from step 2 is subsequently paired with the current state vector $\mathbf{x}_s$ to form an augmented state $\mathbf{x} = (\mathbf{x}_s, \mathbf{x}_a)$.
4) The new augmented state $\mathbf{x}$ is kernelized in terms of $k(\cdot, \mathbf{x})$ such that any state $\mathbf{x}$ implicitly maps to a function that expects another state $\mathbf{x}'$ as an argument; $k(\mathbf{x}', \mathbf{x})$ evaluates to a high value provided that $\mathbf{x}$ and $\mathbf{x}'$ are strongly correlated.
5) Using (6.11), the value prediction for the new augmented state is given by:

$$\langle Q^+, k(\cdot, \mathbf{x}) \rangle = \sum_{i=1}^{m} \alpha_i \langle k(\cdot, \mathbf{x}_i), k(\cdot, \mathbf{x}) \rangle = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}, \mathbf{x}_i) = Q^+(\mathbf{x}).$$

Note that in 5), we have used the fact that Q^+ is a function in RKHS and thus assumes the reproducing property of the kernel (see Section 5.5). Note also that the sampling process in 2) can be viewed as an integral part of the operation associated with the action operator or alternatively as an external process, both of which lead to the same result. Using the formulation above, the set of particle states in Ω (i.e. $\{\mathbf{x}_i\}$) corresponds to a set of kernelized state-action pairs $\{k(\cdot,\mathbf{x}_i)\}$, each of which is a vector in RKHS for the purpose of value prediction and policy derivation. The value of a new kernelized state-action pair $k(\cdot,\mathbf{x})$ can thus be obtained by comparison with all the existing (kernelized) evidences in Ω, with the value largely determined by correlated particles with relatively higher (fitness) values in terms of the inner product $\langle k(\cdot,\mathbf{x}),k(\cdot,\mathbf{x}_i)\rangle = k(\mathbf{x},\mathbf{x}_i)$. A higher fitness value associated with a kernelized state-action pair implies a higher preference for the corresponding action choice given that the policy is defined in terms of an increasing functional of Q^+; i.e. $\pi\left[Q^+(\mathbf{x})\right]$ or equivalently, $\pi\left[\langle A,k(\cdot,\mathbf{x})\rangle\right]$.

6.7 Policy Inference: An Algorithm

Because the policy learning in the reinforcement field depends on experience particles, a sample update rule is required so that the memory will not exceed a given bound. Several aspects play into the role of constructing a proper mechanism for the particle evolution. First, the new particle retained in the memory needs to reflect the latest dynamics of the system so that the policy generalized from these particles will be as accurate as possible. This can also be explained by noting that the policy is driven by the following two continuous processes in parallel: i) the kernel hyperparameter tuning by ARD and ii) the fitness value prediction by GPR. Recall from (6.7) that the ARD procedure identifies the hyperparameters in a manner that trades off between data fit and model complexity, leading to the best explanation of the data (from the perspective of maximizing the marginal likelihood). Thus, a periodic update using ARD is a key

mechanism for the reinforcement field to keep up with the latest utility signals from the underlying control learner (to be discussed further). The process ii) above is just the flip side of the same coin in that the current fitness values referenced by the experience particles in memory effectively assimilate the update history of the utility estimates from the underlying control learner. Thus, the prediction for a new augmented state will only be as good as the hyperparameters.

On the other hand, since the goal of policy learning is to steer the agent towards a state sequence leading to a maximum utility (as a global constraint), it is imperative for the agent to retain the particles aligned with that objective. This is to say that in each step of the decision process, the action operator should more often than not resolve to a localized policy that triggers the state transition (as a local constrain) to follow the path in the direction of increasing fitness value. Note that the policy inferred from the action operator at a given augmented state, $(\mathbf{x}_s, \mathbf{x}_a)$, is locally applicable to all the neighboring states as long as they are within the scope of a given correlation degree measured by the kernel. This is a property of policy inference in the reinforcement field in which any given two particles are considered similar as long as the combined effect of the state and action ends up being similar (Figure 6.3). Yet from another perspective, this policy generalization property is indeed desirable because the state transition, given a fixed state and action, will vary over time, depending on the action parameters. The random variations in the action parameters express the uncertainty in the action dynamics, which can either be interpreted as the uncertainty inherent in the action performance or the uncertainty from the environment. We shall discuss more of the equivalency of stochastic effects inherent within the (observed) state or the action and its implications in Section 6.7.2. Moreover, depending on the exploration strategy of the policy search, the agent can be chosen to always follow a stochastic policy or a greedy policy after sufficient exploration of the state space. In the empirical study at the end of the chapter, we will consider a softmax exploration strategy in (6.8) that gradually reduces to an ε-greedy policy.

6.7.1 Experience Association and Particle Evolution

In order for the agent to have a means to "interpolate" the best policy given the experience particles, we shall now introduce a central operation for the particle update – experience association. Additionally, two attendant definitions are given: i) hypothetical state and ii) particle polarity.

Definition 6.5 (**Hypothetical State**) *A hypothetical state* s_h^+ *is an augmented state formulated by the agent at a given state* $s \in S$ $(s = \mathbf{x}_s)$ *replicating the primitive action* $a \in A$ $(a = \mathbf{x}_a)$ *from an experience particle* $\omega \in \Omega$ *to give* $s_h^+ = (\mathbf{x}_s, \mathbf{x}_a)$.

Definition 6.6 (**Particle Polarity**) *Positively-polarized particles (or positive particles) are those associated with a non-negative temporal difference* $(TD \geq 0)$ *whereas negative particles are associated with a negative temporal difference* $(TD < 0)$.

The operation of experience association is inspired by a basic learning principle: if a decision had led to a desired result in the past (in terms of the fitness value), then the experience can be reused for similar situations. An example can be observed in Figure 6.3 where the ideal action at query points A, B and C are aligned with the control policy embedded in their neighboring particles. Whether two particles are (sufficiently) correlated is again determined by the kernel. Formally speaking, the experience association operation is performed by the agent at a state $s = \mathbf{x}_s$ associating an instance of the past memory encapsulated in a particular experience particle $\omega \in \Omega$. The agent subsequently "extracts" the action $a = \mathbf{x}_a$ from the particle to form a hypothetical state $s_h^+ = (\mathbf{x}_s, \mathbf{x}_a)$ (Definition 6.5). The resulting new state s_h^+ is then compared with the particle referencing an augmented state $s^+ = (\mathbf{x}_s', \mathbf{x}_a)$ using the given kernel k; that is, by evaluating $k(s_h^+, s^+)$ where $s_h^+ = (\mathbf{x}_s, \mathbf{x}_a)$ and $s^+ = (\mathbf{x}_s', \mathbf{x}_a)$. If

$k(s_h^+, s^+)$ goes above a prescribed threshold τ, i.e. $k(s_h^+, s^+) \geq \tau$, then s_h^+ is said to be (strongly-)correlated to the augmented state s^+. Thus, the threshold τ effectively determines the size of the shaded circles in Figure 6.3. By applying experience association, the agent is given a higher preference for the policy embedded in a neighboring particle.

Next, in order for the agent to distinguish between a preferable action and undesirable ones, this dissertation adopts a dichotomy by labeling particles in terms of the temporal difference (TD) obtained from the underlying control learner (Definition 6.6). Recall from (4.10) that a non-negative TD (i.e. $R(s,a) + \gamma Q(s',a') \geq Q(s,a)$) implies an improvement in the utility estimate and thus that the corresponding action is in general preferable to those with negative TDs. Thus, the particle polarity can be used as a heuristic for the particle evolution.

As mentioned earlier, in order for the action operator to resolve a state transition along the direction that increases the fitness value, the older particles are to be replaced by the new particles with relatively higher fitness values. However, care must be taken in such replacements. The new experience particle subsumes an older one only when they are sufficiently correlated. For a particle update to occur, first the polarity must be aligned: particles with similar states (e.g. nearby positions in a navigation domain) but with an opposite polarity are indicative of a high dissimilarity in actions. This gives rise to the particle update rule in Figure 6.4, referred to as *particle reinforcement* in later discussion because such an update rule promotes a stronger reinforcement field by preserving particles leading to higher fitness values. Note that the particle update rule in the particle reinforcement algorithm assumes that initial fitness values are not overestimated. Only under such condition will the positive particles continue to increase their signal strengths (i.e. fitness values) throughout the learning cycles. Note also that the update rule for positive particles is logically opposite to the rule for negative particles. This is due to the fact that the TD values for positive particles are expected to increase over time as the policy learning proceeds, whereas negative particles have decreasing TD values. The end result of this polarity-oriented update rule is to drive positive particles

function ParticleUpdate(p, k, T, Ω) **return** Ω_{new}

inputs: p, a new particle
$k(\cdot,\cdot)$, a kernel function
T, lower correlation bound
Ω, the current training set (particles)

1. **rank** all the correlated (i.e. $k(p,\omega \in \Omega) \geq \tau$) and polarity-aligned particles in the order of their correlation degrees from **high** to **low** to give Ω', where $|\Omega'| \leq |\Omega|$
2. **foreach** ω in Ω' :
   2.1 **if** $TD(p) \geq 0$ // rule for *positive particle*
   then replace ω by p iff ω has a smaller fitness value
   2.2 **if** $TD(p) < 0$ // rule for *negative particle*
   then replace ω by p iff ω has a larger fitness value
   2.3 **if** a match found, **then break** // *done with update*
3. **return** Ω_{new} // $\Omega_{new} \neq \Omega$ *iff replacement occurred*

Figure 6.4 Particle Reinforcement: a particle update rule that preserves "good" particles that align with the objective for obtaining higher accumulated reward while also retaining the "bad" particles as counter examples by which the agent will avoid repeating the same mistakes.

towards a better policy while negative particles remains as counter-examples throughout the whole learning cycle. The iteration for the particle replacement (Step 2 in Figure 6.4) terminates as soon as a match is found. Conversely, no update will occur if no match is found among the current population in memory, in which case the new particle is simply discarded. Note that the update rule can be flexible in the sense that an exploratory strategy can be introduced to the particle update criteria (Step 2.1-2.2). For instance, simulated annealing strategy can be applied to the non-matching particles particularly in the beginning of the policy learning cycle. Nonetheless, when the size of the training set Ω is large, a global search for a matched particle can be expensive. The computational cost can be alleviated by partitioning the state space with each partition referencing a fixed number of positive and negative particles. In this manner, only a local search is required given that the particles in the same state partition are guaranteed to be more correlated than the others to the new particle referencing a state within the partition boundary.

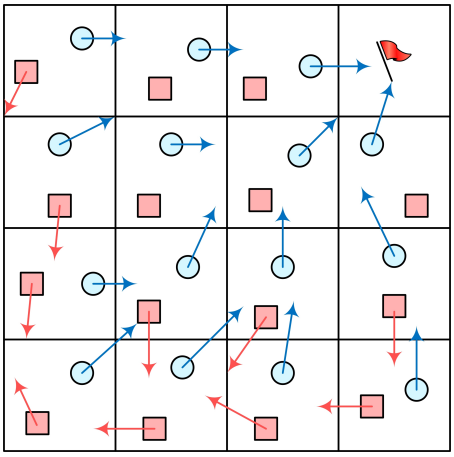


Figure 6.5 A navigation domain where the agent seeks a path to retrieve the flag following the shortest route.

Figure 6.5 illustrates a navigation domain in which the agent aims to travel from the start position to the goal area following a shortest path. The start position is marked by a cross in the lower left square and the goal area is enclosed by the upper right square with a flag. The agent has the freedom to move in all directions within the boundary of the continuous state space (i.e. borders marked by thick lines). The allowable actions include the 12 movements parameterized by a radius and an angle increment $(\Delta r, \Delta\theta)$ adopted earlier in Figure 6.2 and 6.3. Further, the state space is partitioned into a 4-by-4 grid for the purpose of estimating utility values associated with each state area as an approximation to a continuous-state MDP. As mentioned earlier, SARSA is used as the underlying control learner in order to obtain utilities as guiding signals for estimating a fitness value for each particle (using GPR). In this example, each state partition is configured to reference only two particles with one being positive and the other negative. Positive particles are depicted by small blue circles, each of which points to a relatively preferable direction to move. By contrast, negative particles are represented by the small red squares. To keep the figure uncluttered, only selected negative particles are marked with arrows indicating their navigational policies. Each particle represents a policy indicated by

RF-SARSA(k, M,T, Ω_0):

// k: *The kernel function with hyperparemters* $\boldsymbol{\theta}$ (e.g. (6.3))
// M: *parametric action model with n choices*: $\{a^{(i)} \in A \mid i = 1 \sim n\}$
// T: *period for ARD procedure*
// Ω_0 : *initial source of particles (or simply an empty set)*

1. Evaluate the initial hyperparameters $\boldsymbol{\theta}$ via ARD using Ω_0 or simply start with an initial prior $\boldsymbol{\theta}_0$.
2. Initialize $Q(s,a^{(i)})$ arbitrarily.
3. Initialize particle set (i.e. training set for GPR): $\Omega \leftarrow \Omega_0$.

Repeat (for each episode)

4. Initialize s.
5. Given a state s, evaluate Q^+ for each $a^{(i)} \in A$; i.e.
   foreach $a^{(i)} \in A$:
   5.1 $a^{(i)}$ resolves into the value $\mathbf{x}_a$ with $s \leftarrow \mathbf{x}_s$ according to the yield function model.
   5.2 Evaluate $Q^+((\mathbf{x}_s^{(i)},\mathbf{x}_a^{(i)}))$ using GPR (see (6.2) & (6.5)).
6. **Choose** $a^{(i)}$ from s following the current policy: $\pi(s,a^{(i)}) = \pi[Q^+]$ (e.g. softmax in (6.9)) using Q^+ values from step 5

Repeat (for each step in the episode):

7. Run **ARD** only every T steps
   7.1 (Optional) $T \leftarrow T + f(T)$ // *gradually increase T*
8. From s **take action** $a^{(i)}$, and observe r and s'
9. **Generate** a new particle p referencing the augmented state $(\mathbf{x}_s,\mathbf{x}_a^{(i)})$ obtained from step 8
10. Run **ParticleUpdate(*p*)** // *add p to* Ω *or discard p*
11. **Choose** $a^{(j)}$ from s' using $\pi[Q^+]$ (see step 5 and 6)
12. **Update** $Q(s,a^{(i)})$ using Equation (4.10)
13. $s \leftarrow s'$; $a^{(i)} \leftarrow a^{(j)}$

Until s is terminal

Figure 6.6 RF-SARSA.

the associated augmented state and its fitness value. Through the property of the kernel with (6.11) and the GPR prediction in (6.5), all the red and blue particles combined establish a particular reinforcement field, representing a policy generalization mechanism surrounding the neighboring state of the particles. Note that two conditions limit the range of the motion given an action choice: i) allowable range in action parameters ii) the state space boundary. Recall that

an action operator effectively incorporates both the state and action information and always corresponds to a fitness value given that all the parameters are fixed. Thus, during the course of navigation, if the action operator resolves into a position and a motion that leads outside of the state boundary, then the action will only be executed up to the boundary; that is, the agent will move as far as possible before hitting the border and remain there until the next decision step. This is illustrated by the dotted circle enclosing two negative example particles (in red squares), each of which embeds a motion that eventually hits the border. Illustrated in Figure 6.6 (on previous page) is an example algorithm in the reinforcement field using SARSA as the baseline learner. The policy learning method using the property of the reinforcement field is prefixed by RF-, giving rise to the name RF-SASAR.

6.7.2 Soft State Transition

Action operators, similar in a way to the primitive action set in standard RL, are chosen to represent only the local information compared to the scale of the state space. In a standard MDP, executing a primitive action triggers a state transition in which the action can be considered as one of the variables in addition to state features that may vary across states. With action operators, the concept of state transitions is extended further to the transition between pairs of correlated augmented states, each of which combines the state vector and the action vector with their feature values determined only upon their merging. As mentioned earlier, the uncertainty captured by action parameters can be explained using an equivalent model over the stochastic observation of the state. The top of Figure 6.7 conceptualizes the dual property of the uncertainty in the action and in the corresponding start state adapted from the navigation domain from the earlier figures. Similar to Figure 6.3, a dot (i.e. a state) combined with an arrow (i.e. an action) represents a local policy referenced by an experience particle. The uncertainty modeled by action parameters results in, for instance, two of the possible motions on the left (in Figure 6.7) while equivalently on the right indicate two

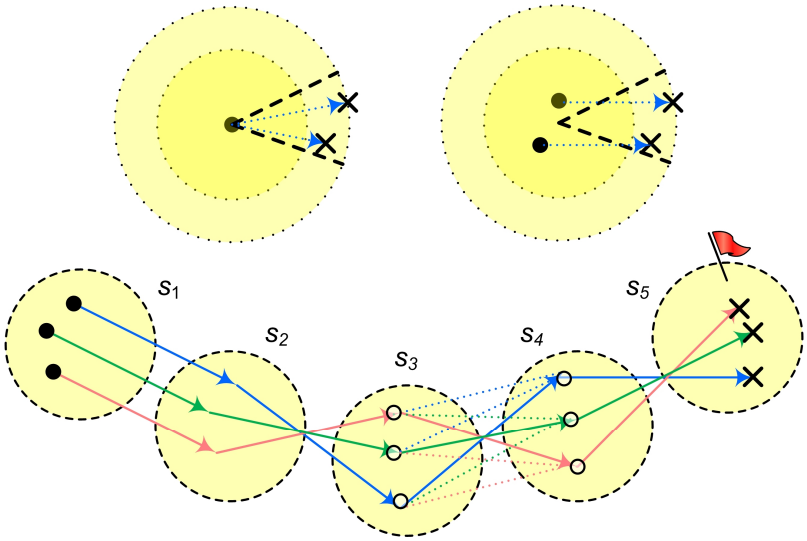


Figure 6.7 Soft state transitions: the top two concentric circles compare two equivalent one-step transitions with uncertainty captured by action parameters (left) and observed states (right), respectively. Enclosed within the 5 "soft states" below in circles are 3 out of 3^4 possible concrete transitions starting from the leftmost soft state to the rightmost state with a flag, all of which correspond to the same soft state transition.

deterministic actions (i.e. action 3 in Figure 6.3 with uncertainty removed via averaging) applied to two nearby states that lead to the same successor states.

The bottom half of Figure 6.7, on the other hand, illustrates 3 trajectories (all starting from the leftmost circle) consisting of "concrete state transitions," all of which lead to the same goal area enclosed by the rightmost circle (s_5) marked by a flag. On a larger scale, however, all of the trajectories approximately follow the same route in terms of the sequence of the 5 successive circles from s_1 to s_5. Each circle has a non-zero volume as opposed to a concrete state (essentially a point in the state space) and thus is referred to as a *soft state* in this dissertation. The state transition by virtue of correlation under experience association effectively relaxes the start and end points for each transition; namely, as long as the state and action vector are jointly correlated in terms of the kernel measure (i.e. $k(\mathbf{x},\mathbf{x}') \geq \tau$), their combined effect will lead to approximately the same state region enclosing yet another set of highly-correlated augmented states with correlated fitness values. In other words, a soft state transition

as such bridges any two sets of highly-correlated states with their volumes effectively controlled by the threshold parameter τ in the kernel-induced feature space. The soft state boundary will remain identical in the entire state space for a stationary kernel that is translation invariant (e.g. SE kernel in (6.3), Matérn class kernels [14], etc); however, the boundary will vary in the state space with a non-stationary kernel such as the dot product kernel:

$$k(\mathbf{x},\mathbf{x}') = \sigma_0^2 + \mathbf{x}^T \Sigma \mathbf{x}', \tag{6.13}$$

where Σ represents a covariance matrix for the input vector components. In Figure 6.7, the 3 hollow dots from within the soft state s_3 reference 3 out of 9 possible concrete state transitions to the corresponding hollow dots in the soft state s_4; however, the 9 state transitions can all be captured, at a higher level, by the same one-step soft state transition (albeit with less accuracy).

6.7.3 A POMDP Analogy

The soft state transition through experience association can also be achieved via a probabilistic model that identifies an unexplored yet correlated (augmented) state by drawing an analogy to the belief state formulation in POMDPs [93, 94]. Recall from Section 4.1.1 where the tuple $\langle S,A,T,R \rangle$ effectively represents an MDP. For an environment where states are only partially observable, two additional elements modeling state observations are incorporated into the representation, giving rise to the POMDP extension with the tuple $\langle S,A,T,R,O,Z \rangle$. In particular, O represents the observation space and Z represents an observation function in terms of the conditional probability $Z(s',a,o') = p(o_t = o' \mid s_t = s', a_{t-1} = a)$ evaluated in parallel to the transition probability $T(s,a,s') = p(s_t = s' \mid s_{t-1} = s, a_{t-1} = a)$. That is, while T specifies the probability of transitioning from the state s to s' by taking the action a, Z specifies the probability of receiving an observation o at s' in response to the same state transition. Due to the issue of partial accessibility of the state, the transition function T alone can no longer be used to specify the agent's worldview in regard to the state transition. Instead, the agent

maintains a probability distribution over the (true) state given an often incomplete measure of the state; namely, an observation. The observation model Z combined with the transition model T is then used to draw a conclusion of the most probable next states in terms of a conditional probability distribution, which gives rise to the notion of a belief state. A belief state is typically modeled by a conditional probability density of the current state given the history; i.e.

$$b_t(s') = p(s_t = s' \mid I_t), \tag{6.14}$$

where I_t denotes the history of the observed states up to the time step t. In particular,

$$I_t = \left(o_0, o_1, ..., o_t, a_0, a_1, ...a_{t-1}\right),\ t = 1,2,...$$

$$I_0 = o_0$$

By applying Bayes' law and subsequently the Markovian property over the observations (and states), (6.14) can be greatly reduced to:

$$b(s_t = s') \propto p(o_t \mid s_t, a_{t-1}) \int_S p(s_t \mid s_{t-1}, a_{t-1})\, b_{t-1}(s_{t-1} = s) ds_{t-1} \tag{6.15}$$

Note that (6.15) omits the normalization factor in the denominator for simplicity and, in contrast to (6.14), does not depend on the past observations I_{t-1} including the past action sequence. Note also that by Markov assumption over the state transition, the current history at time t does not depend on the past state sequence $\left(s_0, s_1, ..., s_{t-1}\right)$; thus, these states are not included in I_t as well. By simple comparison, the RHS of (6.15) is equivalent to $Z\left(o', a, s'\right) \int_S T(s, a, s') b(s) ds$ using the definition of the state transition function T and observation function Z.

Inferring the current state using the belief-state model is analogous to using experience association over the experience particles represented through the functional data set $\Omega : \left\{(\mathbf{x}_1, q_1), ..., (\mathbf{x}_m, q_m)\right\}$. In particular, by comparing (6.5) and (6.11), the weights, α_i, in (6.11) can be re-expressed in a vector form: $\boldsymbol{\alpha} = K(X, X)^{-1}\mathbf{q}$, making it explicit that Q^+ is a linear combination of kernels centered on augmented states. Alternatively, by setting

$h(\mathbf{x}_*) = K(X,X)^{-1}\mathbf{k}(\mathbf{x}_*, X)$, the fitness value function (6.11) can be expressed explicitly as a linear combination of predicted signals (i.e. fitness values):

$$Q^+(\mathbf{x}_*) = \mathbf{k}(\mathbf{x}_*, X)^T K(X,X)^{-1}\mathbf{q} = h(\mathbf{x}_*)^T \mathbf{q}, \tag{6.16}$$

where $h(\mathbf{x}_*)$ is also known as the *weight function* in the context of kernel regression [38]. From the form of $h(\mathbf{x}_*)$, it is not difficult to see that its value depends on the data but not on the prediction; that is, $h(\mathbf{x}_*)$, is driven only by the localized policies expressed through the kernelized augmented state $k(\cdot, \mathbf{x}_i)$ (where $\mathbf{x}_i$ is the *ith* column vector of *X*) but not by their fitness values (q_i). If the particles in the x-space become so dense that $h(\mathbf{x}_*)$ approaches a continuous curve, then $h(\mathbf{x}_*)$ can be analyzed in terms of the power spectrum of $k(\cdot,\cdot)$ and is also referred to as the equivalent kernel (EK) by an analogy to *kernel smoothing* [37, 38]. The EK generally exhibits a localization property with its value higher in the neighboring points relative to the test point and gradually decaying with distance. The localization property becomes more prominent as the number of training points increase. For instance, the EK for the SE kernel in (6.3) can be approximated by a sinc function [37]. By (6.16) and the continuity assumption of the augmented state space (i.e. the x-space), $h(\mathbf{x}_*)$ as the EK effectively encodes the correlation degree of the query state $\mathbf{x}_*$ to all the other particles kept in the memory where $\mathbf{x}_*$ effectively serves as the "center" for the evaluation. This again justifies that the fitness value prediction by (6.16) will weigh heavier on more correlated evidences. Although the localized EK suggests its use as a kernel density estimator for building a probabilistic model for comparing experience particles, it is made complicated by the fact that EK generally has a higher spatial frequency than the original kernel $k(\mathbf{x}_*,\cdot)$ for points away from the center. That is, while $k(\mathbf{x}_*,\cdot)$ is always non-negative, the curve of the EK has zero-crossings similar to the side lobe of the sinc function. Nevertheless, in the case where $k(\cdot,\cdot)$ is also a localized kernel (e.g.

SE kernel), an adaptation to the belief state model in (6.15) can then be achieved by reusing the associated kernel expressions in $h(\mathbf{x}_*)$ to define the weight:

$$w_i = \frac{k(\mathbf{x}_*, \mathbf{x}_i)}{\sum_{i=1}^{m} k(\mathbf{x}_*, \mathbf{x}_i)} \tag{6.17}$$

By (6.17), the fitness value function can alternatively be predicted through a linear combination of these weights: $\hat{Q}^+(\mathbf{x}_*) = \sum_{i=1}^{m} w_i q_i$. This is also known as the Nadaraya-Watson model [14, 38]. Because of the constraint: $\sum_i w_i = 1$, each weight w_i can be endowed with a probabilistic semantic that expresses the probability of the test point $\mathbf{x}_*$ lying within the boundary of the soft state referenced by the existing augmented state $\mathbf{x}_i$ assuming that their action vectors are approximately identical. That is, a high probability evaluated by (6.17) effectively indicates that executing the local policy embedded within $\mathbf{x}_*$ will very likely lead to a correlated result referenced by the old experience $\mathbf{x}_i$. If, however, the underlying state vectors associated with $\mathbf{x}_i$ and $\mathbf{x}_*$ are sufficiently close within the reach of an action, then a high probability by (6.17) also implies their spatial proximity in the state space. Policy interpolation in this manner indeed is reminiscent of that in Figure 6.3 where the control policy for a new point is determined largely by the policy referenced by its neighboring particles. Note that (6.17), similar to (6.14), implicitly models a conditional probability density over the learning history due to the fact that the hyperparameters, shared by each kernel term $k(\mathbf{x}_i, \mathbf{x}_*)$, are optimized via the ARD process over the functional data set Ω, which essentially assimilates the past learning history. Consequently, the density estimation using (6.17) also depends on the past experience through the optimized hyperparameters. As we shall see shortly, the experience association and the soft state transition are the foundation to further compressing the representation required for policy learning. However, the experience association used in this dissertation, for simplicity, will adopt the deterministic scheme mentioned earlier by using the threshold parameter τ. Further analysis of the probabilistic model and its empirical study are left to future research.

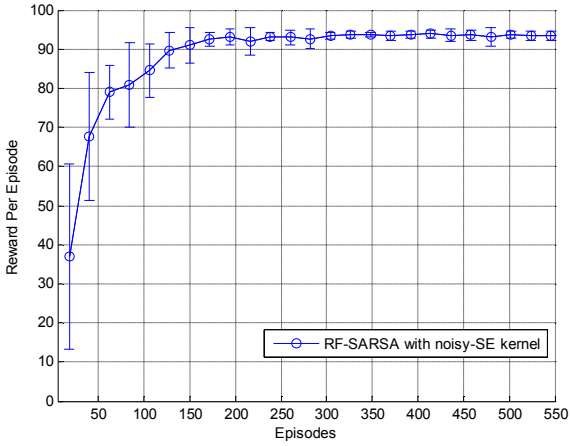


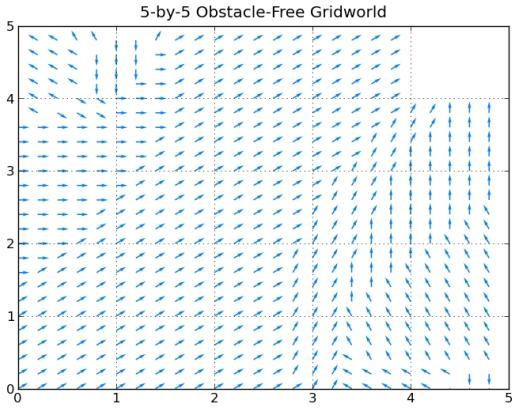
Figure 6.8 RF-SARSA with noisy SE kernel.

6.8 Empirical Study

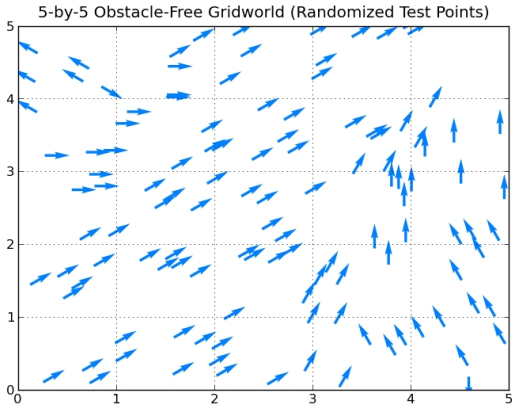
This section demonstrates RF-SARSA by reusing the navigation domain in Figure 6.5 with different configurations and layouts. In the first experiment, we will study the simplest scenario where there is no obstacle in the state space such that the agent is free to move around in all directions. The state space is partitioned into 25 evenly-spaced grids, each of which is configured to reference 3 positive and 3 negative particles respectively, leading to a maximum of 150 particles to maintain in memory. Since the goal is for the agent to learn the policy following a shortest path (see also Figure 6.5), a cost (-1) is imposed per step of travel whereas a reward (+100) is given upon reaching the goal area. Assuming that each side of the environment is of 5 unit length, the action parameter Δr is set to have a target value of 1.0 unit length but varies between -0.8 to +1.2 unit length modulated by a Gaussian noise, applicable to all action choices. Equivalently, this is to model the radius-specific variation w_i in (6.1) (i.e. the yield function model) in terms of this constrained Gaussian noise. The angular parameter $\Delta\theta$ takes on 12 equal partitions around the circle (see Figure 6.2 and 6.3) resulting in 12 different target angles for the actions, each of which has ± 15 degrees of tolerable error. Following a shortest path to the goal area will obtain a reward of approximately 94. Figure 6.8 shows the

performance of RF-SARSA with the SE kernel in (6.3). Using the product kernel in (6.4) leads to a similar learning curve and thus is not included in the Figure 6.8 to keep the figure uncluttered. The comparison for the policy generalization using different kernels, however, can also be achieved by a careful analysis in their associated reinforcement field plots. The field plot helps in visualizing the control policy for arbitrary points in the state space that can be inferred from the known particles retained in memory.

Figure 6.9 illustrates the field plot induced by the noisy SE kernel in the obstacle-free, 5-by-5 grid world where the agent starts from an arbitrary location from within the lower left grid area and travels to the upper right grid representing the goal area, which is similar to the setting in Figure 6.5. In particular, Figure 6.9 (a) depicts the case where the test points (in the state space) are evenly spaced within each state partition (in the grid form) whereas in Figure 6.9 (b), the test points are randomly generated for each state partition. Arrows in the field plot indicate the direction of travel reflecting action preferences at spatially different areas. For instance, the state partitions in the diagonal are in favor of (stochastic) actions 1 and 2 (see Figure 6.3) that tend to resolve into the navigational direction aligned with the shortest path. Notice that both the uppermost and rightmost state partitions (especially those near the corners) reference some local control policies that are not consistent with the shortest path (in terms of misaligned arrows). This is due to the fact that the policy search converges prior to a thorough exploration of the state space. Indeed, this is desirable in that the agent is expected to identify an ideal control policy without the need to traverse the entire state space in order to shorten the learning time. For the purpose of illustration, the size of the arrows is scaled according to the size of the grid and density of the test points. As a result, the length of each arrow does not correspond to the actual step size even though each arrow does represent a sampled navigational direction in accordance with the (stochastic) action choice. Due to the simplicity of this domain, the field plots induced by the noisy SE kernel and the product kernel approximately follow the same pattern. As we shall see, in the next gridworld with obstacles lying along the navigational path,



(a)



(b)

Figure 6.9 Reinforcement field plots for the 5-by-5 gridworld with no obstacles. The control policy is represented by the arrows pointing in the preferable navigational direction at spatially different locations. The arrows in (a) are evenly distributed within each state partition whereas those in (b) are randomly generated.

the policy generalization property between the noisy SE kernel and the product kernel begins to exhibit subtle differences.

In particular, imagine the scenario where the obstacle can cause permanent damage to the agent and thus needs to be avoided by all means (e.g. deep craters on a planetary surface suggested by Figure 4.2). This is in contrast to the state space boundary which forces the agent to only navigate within an allowable area. Under this assumption, it is reasonable to assign a

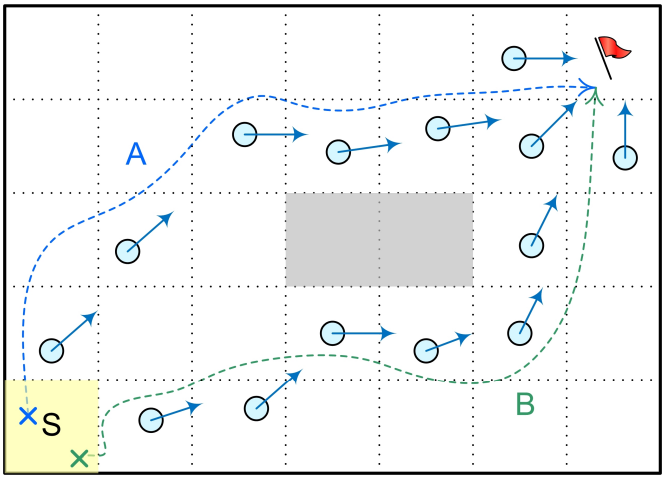
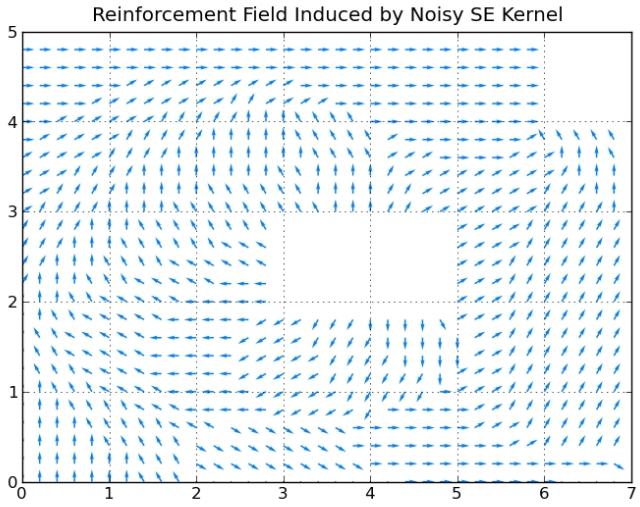
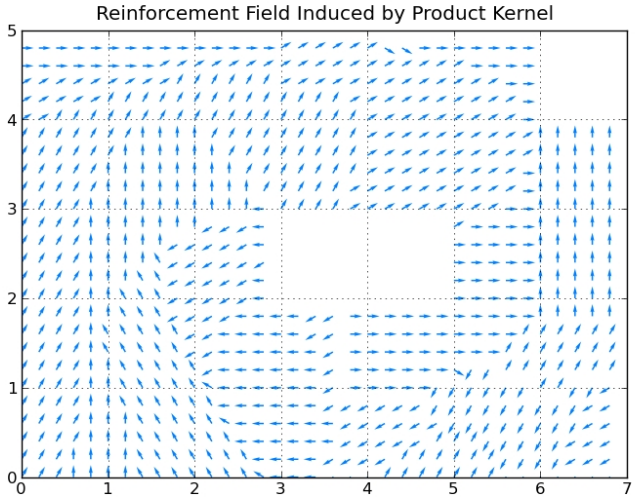


Figure 6.10 A 7-by-5 gridworld with areas filled with obstacles near the center zone. Curves A and B represents two sample paths toward the goal that circumvent the obstacle while minimizing the travel distance as much as possible.

large negative reward (-100) whenever the agent hits an obstacle whereas no penalty is given in addition to the per-step cost when the agent touches the border of the navigation area. The environment setting is depicted in terms of a 7-by-5 gridworld in Figure 6.10 where obstacles are represented by two shaded state partitions (colored in gray) near the center area. Two possible (sub-)optimal routes toward the goal are illustrated in blue and green dashed curves, both of which attempt to circumvent the region with obstacles. Figure 6.11 (a) and (b) illustrate the corresponding reinforcement field established by the noisy SE kernel and the product kernel respectively. As can be seen, the product kernel expresses the greedy policy toward the shortest path relatively better than the noisy SE kernel in the neighborhood of the obstacle (under the same experimental settings including the number of training episodes). However, the policy induced by the SE kernel has a higher guarantee in escaping from the obstacle. For practical purposes, the ability to avoid obstacles may outweigh the benefit of following the shortest path in this scenario. Since actions are constantly influenced by random effects associated with possible errors in the actuator or unexpected environmental shifts, an “optimal action” does not necessarily resolve to an optimal real-time behavior. For instance, choosing the



(a)



(b)

Figure 6.11 Reinforcement field plots for 7-by-5 gridworld with 2 center zones filled with obstacles. The field plot in (a) corresponds to the policy associated with the noisy SE kernel while (b) corresponds to the product kernel. The control policy in (a) suggests relatively smoother streamlines and tend to escape from the obstacle in a more consistent manner.

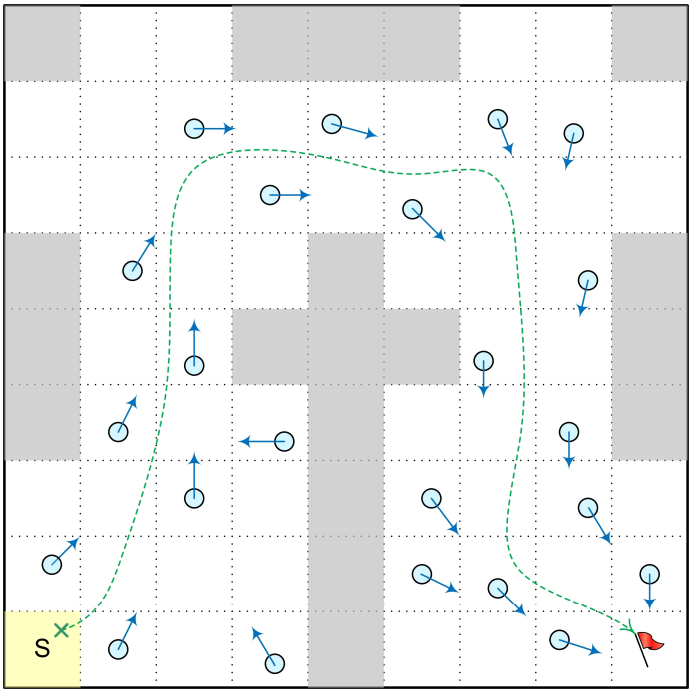


Figure 6.12 A 9-by-9 gridworld with obstacles all over the state space areas. Similar to the case with 7-by-5 state partitions, the green curve represents an example path guided by the distributed experience particles.

action that navigates to the east (i.e. action 3 in Figure 6.3) along the southern border of the obstacle-filled region in Figure 6.10 may not be desirable in practice despite the fact that a successful execution of such an action sequence may lead to a shorter path. Yet, the success rate for such an action sequence is in general very low unless the action outcome always meets the expectation. Similar to the reinforcement field in the obstacle-free gridworld in Figure 6.9, there are relatively unexplored areas in the state space (e.g. the southeast corner) where the policy is not consistent with a shorter path due to the lack of evidences that support a proper policy generalization in these areas. A more complex scenario with relatively more obstacle-filled regions is illustrated in Figure 6.12 in which the agent is required to take a detour in order to reach the goal area. The corresponding reinforcement field plot using a noisy SE kernel is given in Figure 6.13. Similar to the cases we have seen earlier, the state space areas further away from the optimal path (represented by the streamline implicitly defined by the field) are

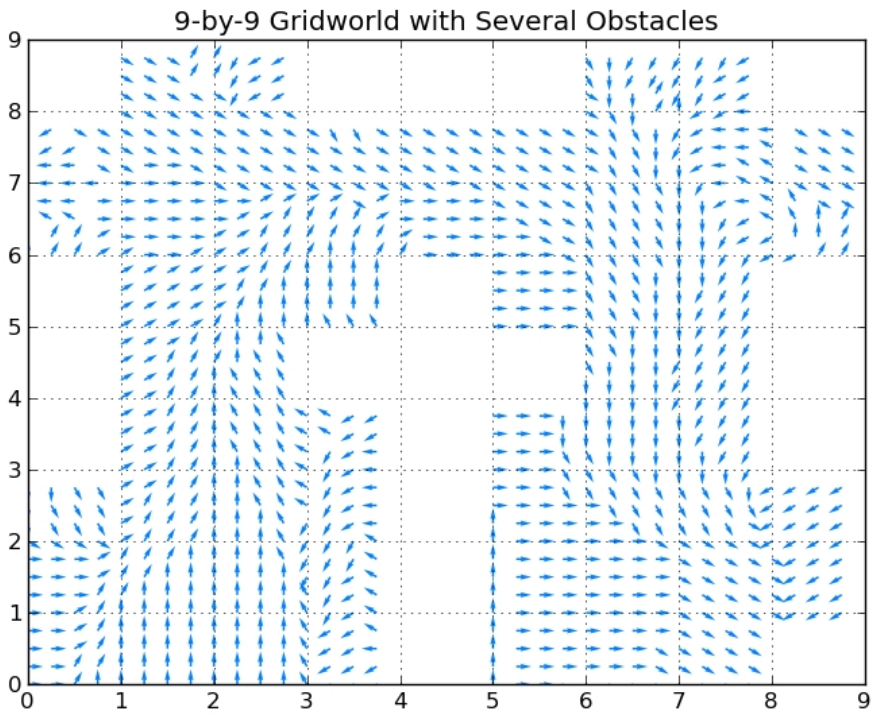


Figure 6.13 Reinforcement field plot for a 9-by-9 gridworld with obstacles that force the agent to take a detour to reach the goal.

likely to be more "distorted" due to the lack of exploration before RF-SARSA converges. Examples are the state space areas near the spatial coordinates (0.5,6.5), (0.5,7.5), and (8.5,7.5) in Figure 6.12. Nevertheless, within these relatively unexplored areas, the agent still gives preference to escaping from the obstacle albeit not necessarily along an optimal path due to a lack of well-trained experience particles as supporting evidences.

Note that standard SARSA can be considered a special case of RF-SARSA by removing the parametric action model and the extra layer of fitness value estimation, thereby reducing the action operator to an ordinary action set with predefined behaviors. Nevertheless, doing so effectively drops the policy generalization capacity and more often than not, more features need to be engineered into the state space to cope with potential complications from action dynamics, leading to an exponential growth in the state space. The need to increase the

state feature due to the stochastic action dynamics will become apparent in the task-assignment domain under time-varying computational resource induced by distributed pilots in the PanDA-PF WMS introduced in Chapter 2. In particular, an increase of pilot-resident servers as matching candidates for the incoming tasks amounts to an increase of task-assignment actions whose stochastic effects are in direct relation to the feature set characterizing these candidate servers. By the formulation of standard RL, this also implies that the state feature necessary to characterize these newly-emerged servers need to be incorporated into the original state space such that the new task-assignment policy can reflect the dynamic property of these new servers. We shall see an alternative method of modeling the state space in terms of the parametric-action model and the notion of reinforcement field in the next chapter.

6.9 Summary and Future Perspectives

The reinforcement field enables policy generalization through properties of the kernel as a correlation hypothesis that associates the policy-embedded particles distributed across the state space. Each particle effectively holds a representative policy that generalizes into its neighboring state and thus, all particles collectively form a field (see (6.11)). The reinforcement field encodes the knowledge of the environment gathered so far, reflecting the progressive advancement in policy search. Through the kernel working with GPR along with the ARD procedure, the agent's worldview continues on updating, which in turn loops back to adjust the control policy reflecting the shift in the environment dynamics. In addition, an action operator is defined that integrates the local dynamics of state transitions with the (fitness) value prediction that drives the policy learning. In this manner, the agent will always favor a state transition that moves towards a higher utility, eventually leading to a state sequence leading to a maximum accumulated reward – the global objective of the policy learning. We will investigate in Chapter 7 a further generalization in terms of action abstractions. The goal is to identify similar decision experiences in terms of the similarity in their augmented state representation. Similar

augmented states are grouped together by taking into account their corresponding fitness values. In this manner, similar control polices can be clustered to formulate an action-oriented conceptual model such that the agent can derive policies at the level of abstractions to reduce the decision points and in the meantime, identify interesting patterns expressed in terms of the correlated state features and the action parameters. As we shall see shortly, the pattern encoded through correlated state features and action parameters can effectively serve as a mechanism that automatically determines the matching criteria between user tasks and candidate machines in a workload management system. Extracting useful matching criteria in this manner can potentially contribute to higher resource utilization in that manually-prescribed requirement statements (e.g. see Section 2.4 or refer to Condor ClassAds [59, 61] for more details) may not objectively reflect the true property of the user task and candidate machines.

CHAPTER 7

CONCEPT-DRIVEN LEARNING ARCHITECTURE

The previous chapter generalizes the RL framework by extending the notion of the parametric-action model to the action operator that takes on an input state, resolves the stochastic behavior of the action and finally produces the next state. In particular, the stochastic effect of an action is modeled in terms of a constrained random vector where each of the coordinates follows a selected task-dependent probability distribution such that the sampled value along each coordinate (upon merging with a state) will fall within a prescribed range. For this, we use a yield function to model such probabilistic behavior of the action. In this way, the sequence of actions applied in the state space effectively unfolds as a random process. The state transition is thus defined operationally through the action (as an operator) acting on the state and subsequently leads to an augmented state as a byproduct. Recall that the augmented state space is the tensor product over the state space and action parameter space and effectively expresses the pairing of a state and action vector.

The action formulation in this manner also facilitates pattern discovery from within the augmented state space, thereby identifying commonality in localized control policies through the operation of experience association. In particular, through the use of GPR with an adaptive kernel as the covariance function, a reinforcement field is established as a progressively updated vector field in RKHS via a periodic ARD process in response to the ongoing TD learning with the control learner. The particles (that reference augmented states) in Ω are encapsulated within the corresponding kernel expressions (see (6.11)), each of which serves as an element that generalizes across the field for further policy inference over new instances of policy-embedded particles. Since the underlying representation for the reinforcement field is

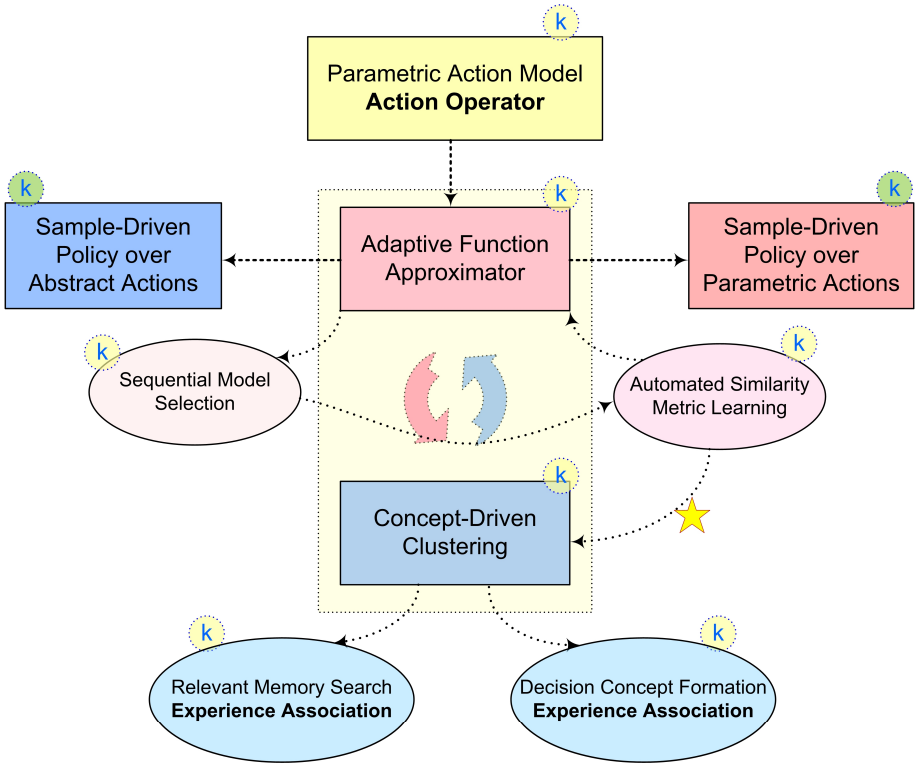


Figure 7.1 The complete generalized RL framework with concept-driven learning architecture (CDLA). The control policy can be learned both at the level of parametric actions as well as high-level abstract actions. The components represented directly through kernel functions are attached with yellow circles whereas both types of the policy functions are indirectly kernelized through their dependency on the value predictor. As such, they are marked by green circles for a distinction. From the figure also find the star that references the key logical flow that enables value function approximation and abstract concept learning to simultaneously occur.

based on the functional data ($\Omega : \{(\mathbf{x}_i, q_i)\}$) with the aforementioned properties, a conceptual model can be developed through the analysis of the functional relation inherent in state features, action parameters, and their corresponding fitness values. Through the functional clustering mechanism and the property of kernel functions, this chapter will complete the remaining logical components colored in blue (of varying hues) in Figure 7.1 to finally achieve abstract policy learning driven by the mutually iterative process between concept-driven action (and state) abstraction learning and the value function approximator depicted in the center (or "torso") of the robot-like schematic.

7.1 Generalized Reinforcement Learning Framework

Recent work in function approximation [2, 3, 5], hierarchical reinforcement learning (HRL) [18, 19], and policy abstraction [15], address RL scaling and adaptability from seemingly different perspectives, but all reduce to two major directions: i) adaptive policy learning ii) compact policy representation. As we shall see later in this chapter, these methods can be further unified into a single piece as a generalized reinforcement learning framework. Function approximation methods have been used extensively to ameliorate the dimensionality barrier in representing the state space of complex systems by using a selective set of basis functions and their linear combinations to represent the value function. The basic idea of function approximation using a basis set is outlined in Chapter 5. However, challenges arise in choosing the right basis set, the size of which also grows rapidly with the dimension of the state space. The proto-reinforcement learning framework (or Proto-RL) [5, 6] goes further by learning the structure of the basis set through samples of experiences in terms of the eigenfunction of the graph Laplacian encoding the topology of the state space. This framework effectively learns the required value-function representation, out of which the control policy is derived simultaneously. In relation to this learning framework, namely *representational policy iteration*, the previous chapter introduced Gaussian process regression (GPR) as an alternative route in representing the potential function, which is also driven by samples of experiences but without the need to learn the basis functions directly.

On the other hand, establishing a high-level abstraction over the state space or within the policy representation is another approach in scaling the RL method. In particular, Hierarchical Reinforcement Learning (HRL) based on clustering [29] and/or subgoal discovery [19] decomposes the otherwise large state space into different smaller regions where sequences of actions (i.e. the options formulation [34]) are determined for the associated subtasks, each of which constitutes an intermediate step necessary to reach the final goal (hence the name subgoal). The complexity in the policy learning is greatly reduced due to a

reduction of the state space through learning the local policy in each subgoal area individually. Factored MDPs [96, 97], on the other hand, reduce the representational size by expressing states implicitly through a set of random variables. Subsequently, a dynamic Bayesian network (DBN) is established as a compact transition model based on the rationale that the transition of one variable often depends only on a relevant subset of the other variables. The abstraction is thus built upon the structural regularity within state transitions in response to the stochastic effect of actions. Doing so effectively aggregates a set of states that respond to an action in a consistent manner. Additionally, the DBN representation can be further extended to capture the context-specific structure [97] from within the conditional probability table (CPT) that exhibits patterns in terms of the assignment to a subset of variables upon state transitions. That is, certain state-action pairs can only occur in a relevant context in which the transition involving the pair is a feasible choice (effectively with a non-zero probability).

In light of the hierarchical and context-dependent algorithmic constructs from the aforementioned methods, this chapter further establishes a concept-driven learning architecture (CDLA) by developing abstractions over similar *decision contexts*, a terminology used to emphasize its definitions over a set of correlated augmented states and their equivalency in the role of the policy generalization when kernelized (see also Section 6.6). Specifically, through the kernel property, a sample of experience not only encapsulates a state-specific “micro-policy” but also generalizes across the neighboring state space that shares approximately the same policy structure (in terms of kernelized state-action pairs). This policy-homogeneous state area is referred to as a soft state (see Section 6.7.2) and is one of the key properties of the reinforcement field.

The abstraction in CDLA to be developed in this chapter will be action-oriented in a manner that implicitly encodes a set of action-selection strategies, each of which serves as a conceptual unit by which the agent derives the control policy. Although the concept-driven policy is similar in the sense of its hierarchical structure to the policy expressed in HRL methods, the

policy representation is fundamentally different in terms of the formulation of abstract actions. Each abstract action in CDLA represents a set of coherent decision contexts that generalize a group of primitive actions taken in different states that nonetheless led to similar end results by virtue of a utility-like measure (fitness value) as mentioned earlier. The concept-driven action abstraction does not necessarily lead to coherent sets of state sequences with spatially or temporally continuous properties as in other abstract action formulations such as the options framework in the context of a Semi-MDP (or SMDP) [34]. In particular, an option when applied in a particular state partition unfolds in terms of a sequence of actions that lead to a terminal state within the partition boundary. The set of actions drives the agent step by step following a contiguous state area. Conversely, the concept-driven abstract action aggregates a set of policy-embedded particles (referred to as experience particles earlier) that reference similar control policies defined by their correlated fitness values. Thus, each primitive action in the abstraction is not meant to be carried out in sequence like an option in a subgoal area. Instead, the action together with its paired state (referenced by a particle) is used to match against other hypothetical decision contexts involving other possible pairings of states and actions. This is similar in some sense to the SMDP homomorphism framework [35] that builds equivalent yet reduced MDPs out of the structural symmetry and redundancy in the state space, which then leads to a form of policy symmetry. We also note that the policy homomorphism framework [15] explicitly represents such situation-specific control policies that generalize across the state space (or even across similar tasks).

With the operation of experience association established earlier in Chapter 6, the concept-driven abstraction can be shown to reflect the correlation between decisions made across the state space and serve as a mechanism to reduce the decision points per state. This is useful particularly in complex domains with a larger action space that has non-stationary outcomes (e.g. task assignments under the framework of PanDA-PF WMS from Chapter 2).

Moreover, each abstraction as a policy-embedded conceptual unit effectively serves as a simple cognitive model that can be shown to reveal the correlation between the state features and the action parameters. Such functional structure that goes across the state and the action space enables the agent to derive a high-level control policy by associating the past memory within each conceptual unit (rather than over the raw action set). In order to discover coherent patterns hidden in such functional data, this dissertation proposes a simple union of Gaussian process regression (GPR) and spectral clustering as a functional clustering algorithm to be discussed in Section 7.4.

7.1.1 Conceptual Hierarchy of CDLA

As an overview, the construction of CDLA hinges upon the reinforcement field and further extends the following learning paradigms as an integrated framework:

1. *Defining actions collectively as a continuous random process*: Actions are not merely atomic, immutable choices with behavioral details predefined but often in reality, involve a variational process as they act on the states.
2. *Learning by associating past memory*: Decisions made across different states (collectively as a decision context) often carry mutually-dependent relations and thus can be linked together by virtue of their correlated reinforcement signals.
3. *Forming conceptual units over coherent decisions*: Correlated decisions can be further assembled together to represent an action-selection strategy.

Figure 7.2 illustrates a conceptual hierarchy corresponding to the aforementioned learning paradigms built sequentially upon one another. The top row depicts a set of experience particles. Each particle encapsulates a localized control policy represented in terms of a compound state-action vector, referred to as an augmented state. The point-wise, localized

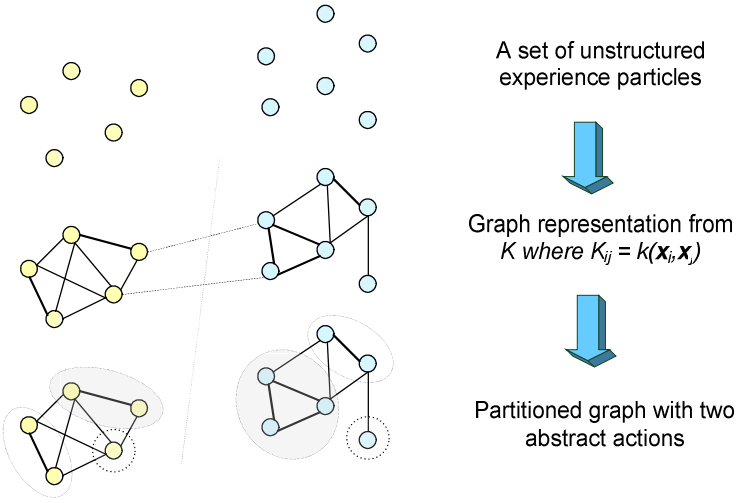


Figure 7.2 Conceptual Hierarchy in CDLA: highly-correlated particles are grouped together to form a coherent conceptual unit in the form of functional clusters at the bottom row. The ellipses within each cluster are the soft states represented through the particles referencing nearby states within the reach of an action.

policy inherent in the augmented state generalizes further across the state space through the use of the kernel function as a correlation hypothesis. Each particle also references a scalar signal – the fitness value – obtained from the utility estimated by the control learner for the state-action pair that led to the corresponding augmented state representation. The particles generated during the course of the policy iteration give rise to a functional data set $\Omega : \{(\mathbf{x}_1, q_1), ..., (\mathbf{x}_m, q_m)\}$, where $\mathbf{x}_i$ represents the particle state as a pairing over state and action vectors and q_i corresponds to its fitness value as a scalar signal. Subsequently, a similarity measure is defined pairwise over the set of particles through the use of the same kernel mentioned earlier, resulting in a similarity graph expressed by the adjacency matrix *K*. Each entry in *K*, i.e. $k(\mathbf{x}_i, \mathbf{x}_j)$, specifies the strength of an edge. A stronger edge corresponds to a higher similarity in the particle state. The bottom row in Figure 7.2 shows the result of partitioning the similarity graph, leading to two conceptual units to be used as abstract actions. Each abstraction effectively encodes a set of coherent localized policies in terms of the correlated particle states along with their correlated fitness values.

Based on these key aspects, this dissertation will conclude with a generalized RL framework that periodically drives three interleaving processes: policy learning, regression, and functional clustering [39, 40] to formulate concept-driven action abstraction. Subsequently, the agent uses abstract actions as decision choices to derive the control policy, further reducing the policy representation from the primitive action space to an often smaller abstract action space. In particular, the notion of the experience association leads to the use of the kernel as a means to gauge the similarity of various decision contexts. Further, all the past decisions in terms of state-action pairs in addition to their fitness value estimates constitute a set of functional data [39]. Finally, to discover coherent patterns hidden in such functional data, GPR and the spectral clustering mechanism works together to formulate the graph partitions shown at the bottom row of Figure 7.2.

7.2 Functional Clustering

GPR identifies the functional structure that relates correlated particles through their correlated fitness values. With this property, it becomes possible to aggregate correlated particles, using the very same kernel as a similarity measure, to form clusters consisting of similar decision contexts. For instance, the 3 query points marked by crosses in Figure 7.3 along with their nearby particles in dots can potentially be grouped into 3 distinct clusters, each of which represents a coherent decision concept. Particles in these 3 clusters reference approximately identical potential values due to the symmetry property with respect to the goal area in this scenario. That is, points A, B, C are approximately at equal distances to the goal and will remain so after the optimal actions are applied (i.e. action 2, 5 and 10 respectively). Given a set of experience particles (i.e. $\Omega : \{(\mathbf{x}_1, q_1),...,(\mathbf{x}_m, q_m)\}$), the goal is thus to identify such symmetry properties within their corresponding localized control policies. Each cluster serves as a conceptual unit representing a guideline in choosing appropriate actions for the states with sufficient correlations. In particular, the policy for a new query state can be inferred by matching

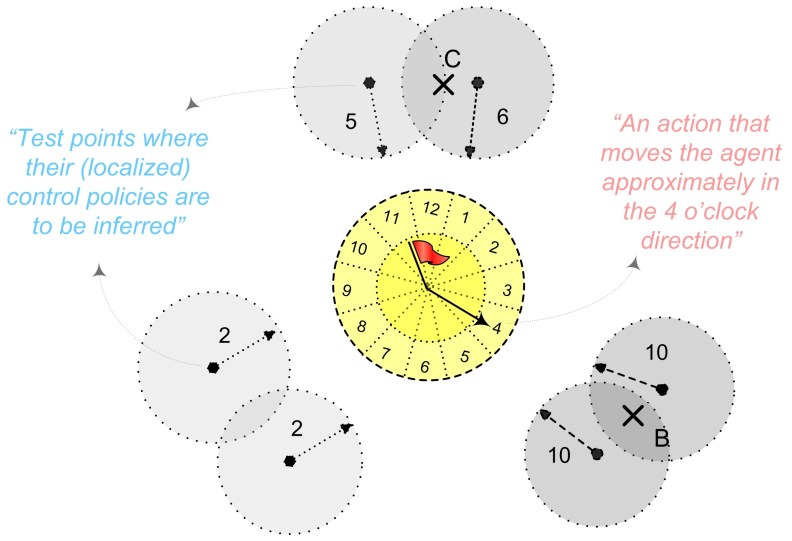


Figure 7.3 Re-adaptation of the navigation domain from Chapter 6: this summarizes Figure 6.2 and 6.3 in which 12 primitive actions are being parameterized by a radius and an angle increment relative to the current position. The task is to navigate toward the center goal area by inferring the motion policies for the unexplored area (e.g. A, B, C) from the neighboring particles.

particles within the cluster via the experience association operation. Recall from Section 6.7 that a set of hypothetical states can be formulated by fixing the action while varying the state vector. Here, in order to find the best match in the cluster, one seeks to find the most correlated particle(s) by comparing the state vector between the query point and the particles while keeping the action vector identical. By this token, the most correlated particle(s) for the query point A in Figure 7.3, for instance, are those nearby particles referencing action 2 as the best known control policy. In other words, by taking action 2 at A, the resulting particle (now referencing a hypothetical state) will become sufficiently correlated to the existing particles, thereby concluding that action 2 must also be the best choice for A. One advantage of clustering correlated local policies as such is to reduce the number of decision points per state when the set of primitive actions is fairly large. More importantly, forming conceptual units over policies also helps to identify the relation between states and actions that jointly lead to similar results. This allows for the agent to develop a cognitive model over decisions in addition to the

decision process itself. As an example, the section of empirical study will present a task scheduling domain in which we shall see that formulating decision concepts is helpful in automatically determining the optimal matching criteria between tasks and servers (i.e. requirement statements [57, 61]).

Moreover, since the clusters can be used as action choices for deriving the control policy (by virtue of experience association), such policy abstraction is action-oriented and can be considered as a type of action abstraction similar to the HRL framework. Also, each abstract action is context-dependent due to the dependency over the underlying states at which actions were applied. Because the desired cluster structure needs to reflect both the correlation in the input data and their corresponding functional responses, the abstraction is accomplished through functional clustering. Functional clustering techniques have been used extensively, for instance, in identifying functionally similar genes from microarray data [43]. Methods of clustering functional data typically aim to "factor" the complex data distribution into a mixture of simpler models such as mixtures of regression models [44], infinite mixture models based on the Dirichlet Process [43], etc. By the rationale of data reuse for saving computational cost, this dissertation simply reuses the covariance matrix evaluated earlier from GPR as an input for the spectral clustering mechanism to form functional clusters. This method is referred to as GP-Spectral Clustering (GPSC) in the later discussion. Note that in the functional clustering procedure, different areas in the state space may share similar control policies as long as the state-action pairs resolve to correlated fitness values as their functional responses.

7.3 The Role of Spectral Clustering

Spectral clustering [24, 25, 28] has emerged recently as a popular method especially for discovering the patterns that are not easily captured by mixture models (e.g. mixture of Gaussians) or k-means due to an unknown yet complex data distribution. The main idea of spectral clustering is to identify cluster structures using eigenvectors of the similarity (or affinity)

matrix derived from the data. Thus, all that is required to define clusters is to have a specific form of pair-wise similarity measures available for the data set. This characteristic also carries over to the case where feature vectors of the data are of varying dimensions, in which case, the mixture models or K-means will fail to apply.

Fortunately, the kernel used earlier as a correlation hypothesis over the augmented states can be reused again as the similarity measure here. The covariance matrix *K* from GPR effectively relates the input augmented states to their corresponding fitness value estimates. Thus, to formulate clusters of correlated local policies, it suffices to identity the cluster structure by reinterpreting the covariance matrix *K* as representing a fully-connected similarity graph with weights evaluated by the same kernel. Note that *K* is a suitable choice also due to the property of being symmetric and positive definite (see also Section 5.3). Since highly-correlated augmented states effectively represent similar decision contexts that have led to correlated fitness values, the goal is thus to identify a set of partitions such that the augmented states in the same partition have relatively higher probability to remain correlated to each other upon their soft state transitions. This also implies that any new particle (associated with a new fitness value) that is highly-correlated to an existing partition will be merged into the same partition with a high probability.

Formally speaking, suppose that the entries in the covariance matrix *K* represent the weights in a fully-connected similarity graph denoted by $G = (\Omega, E)$. The vertex set represents the augmented states (i.e. $\Omega : \{\mathbf{x}_1, \mathbf{x}_2 ..., \mathbf{x}_m\}$) and each edge in *E* represents a correlation degree between any two particles. Here we adopt a notational overlap with the functional data set used earlier for simplicity. Since the correlation of augmented states is evaluated in a manner that also reflects the correlation in their fitness values, the similarity graph *G* implicitly encodes a set of graph partitions with similar local policies. Specifically, as long as at least a subset of the experience particles in the same partition is more correlated to the new particle than those from the other partitions, its cluster affiliation can be determined. As this dynamic process goes on,

each partition will eventually contain sets of correlated particles with similar fitness values; equivalently, these particles have a weaker tendency to escape from the same group according to their local policies (because a local policy is effectively represented by an augmented state, a state-action vector pair). Consequently, the control policy represented through the experience particles in the same graph partition corresponds to a coherent decision concept. For reasons above, the random walk graph Laplacian [26], denoted by L_{rw}, is an ideal candidate for grouping similar experience particles in that L_{rw} effectively encodes a Markov transition matrix. To see this, notice first that one can define L_{rw} in terms of the covariance matrix K as a normalized graph Laplacian: $L_{rw} = D^{-1}(D-K) = I - D^{-1}K$, where the degree matrix D (i.e. $D = diag(d_1, d_2, ..., d_m)$) contains diagonal elements defined as follows:

$$d_i = \sum_{j=1}^{m} K_{ij} = \sum_{j=1}^{m} k(\mathbf{x}_i, \mathbf{x}_j) \tag{7.1}$$

Each diagonal entry in D can be interpreted as the total weights (or correlation degrees) from the *ith* experience particle to all the other particles in Ω. On the other hand, a first-order Markov transition matrix can be defined as $P = D^{-1}K$ such that

$$P_{ij} = \frac{K_{ij}}{\sum_{j=1}^{m} K_{ij}} = \frac{k(\mathbf{x}_i, \mathbf{x}_j)}{d_i} \tag{7.2}$$

Since each row in P sums to 1, P_{ij} can be interpreted as the probability of transitioning between states in $\mathbf{x}_i$ and $\mathbf{x}_j$ by executing their corresponding actions if the states are mutually reachable by the actions. For the cases where $\mathbf{x}_i$ and $\mathbf{x}_j$ are spatially further apart beyond the reach of an action, P_{ij} can still be interpreted as a normalized correlation degree. Note also that for a localized kernel, the formulation in (7.2) is analogous to the Nadaraya-Watson estimator given earlier in (6.17) (see Section 6.7.3).

By (7.2), the relation between L_{rw} and P is apparent given that $L_{rw} = I - P$. In other words, if $(\lambda, \mathbf{v})$ represents the pair of eigenvalue and associated eigenvector for P, then $(1-\lambda, \mathbf{v})$ corresponds to the pair for L_{rw}. As a result, the largest eigenvectors of P or equivalently the smallest eigenvectors of L_{rw} can be used to express the same cluster properties. The rationale behind using eigenvectors as cluster indicators comes from applying *spectral relaxation* to the graph-partitioning problem with the associated graph-based measure (e.g. [25, 27, 28]). In particular, the normalized cut [27] is among the most commonly-used measures and has been shown to have an equivalent relation expressed in terms of the Markov transition matrix [28]. As a simple illustration, first one defines the normalized cut in terms of the total links (or weights) between two graph partitions. Specifically, suppose that we wish to group particles in Ω into *p* partitions; namely, $\Omega : \{\Omega^{(1)}, \Omega^{(2)}, ..., \Omega^{(p)}\}$, where p in general is chosen to be much smaller than the size of the training set *m*. The total weights between $\Omega^{(i)}$ and $\Omega^{(j)}$ can thus be defined as follows:

$$W(\Omega^{(i)}, \Omega^{(j)}) = \frac{1}{2} \sum_{\mathbf{x} \in \Omega^{(i)}, \mathbf{x}' \in \Omega^{(j)}} k(\mathbf{x}, \mathbf{x}'), \tag{7.3}$$

where the factor ½ is introduced such that each link is counted only once. Then, the *p*-way normalized cut can be defined as:

$$NCut\left(\Omega^{(1)}, \Omega^{(2)}, ..., \Omega^{(p)}\right) = \sum_{i=1}^{p} \frac{W(\Omega^{(i)}, \Omega \setminus \Omega^{(i)})}{W(\Omega^{(i)}, \Omega)}, \tag{7.4}$$

where $\Omega \setminus \Omega^{(i)}$ denotes the complement of Ω, and the denominator $W(\Omega^{(i)}, \Omega)$ can be thought of as the sum over all the internal and external links; namely,

$$W(\Omega^{(i)}, \Omega) = W(\Omega^{(i)}, \Omega^{(i)}) + W(\Omega^{(i)}, \Omega \setminus \Omega^{(i)}).$$

Thus, for obtaining *p*-way partitioning of Ω using the normalized cut formulation, one seeks to solve the following optimization problem:

$$\text{minimize } \frac{1}{p} NCut\left(\Omega^{(1)}, \Omega^{(2)}, ..., \Omega^{(p)}\right) \tag{7.5}$$

The optimization objective in (7.5) effectively encourages a balanced graph partition (due to the denominators within the summation of normalized cut in (7.4)) that in the meantime, also minimizes the total escaped links across the partitions (due to the numerators in (7.4)). However, finding the solution to (7.5) is NP-hard [46], thus off-the-shelf spectral clustering algorithms often resort to spectral relaxation. In particular, the normalized cut criterion from (7.5) can be simplified to the following trace maximization problem [27]:

$$\text{maximize } \frac{1}{p} tr\left(Z^T K Z\right), \tag{7.6}$$

where $Z = X(X^T D X)^{-1/2}$ and X is an $m \times p$ matrix with each column as a partition indicator (i.e. a binary-valued vector with an entry 1 representing the belonging to a partition and 0 otherwise). Note that K is the original covariance matrix from which the weights in (7.3) are evaluated and D is the degree matrix defined earlier. A simple matrix multiplication shows $Z^T D Z = I$ as an implicit constraint for (7.6), where I is a $p \times p$ identity matrix. For the solution to be tractable, (7.6) can be approximated by using the top p eigenvectors of $L = D^{-1/2} K D^{-1/2}$ with the constraint $Z^T D Z = I$ relaxed such that, Z as a function of X, is allowed to assume continuous values (instead of the binary integer). Expressing the solution in terms of the eigensystem $L\mathbf{v} = \lambda \mathbf{v}$ and multiplying both sides by $D^{-1/2}$ gives $D^{-1} K D^{-1/2} \mathbf{v} = \lambda D^{-1/2} \mathbf{v}$, which can be written to be:

$$P(D^{-1/2}\mathbf{v}) = \lambda(D^{-1/2}\mathbf{v}),$$

Thus, the top p eigenvectors for the system $L\mathbf{v} = \lambda \mathbf{v}$ also corresponds to those for the Markov transition matrix P but are scaled by the factor $D^{-1/2}$. Lastly, given that $L_{rw} = I - P$, the smallest p eigenvectors thus can be used to compute the a discrete partitioning of Ω. In

particular, this research uses the Shi-Malik flavor of the spectral clustering algorithm [46] with the assumption that the number of action abstractions has already been determined. To automatically determine the number of clusters, interested readers may refer to [110] in which Gaussian-based density estimator is used to model the relevance degree to which each eigenvector contributes to the final clustering using an Expectation Maximization (EM) algorithm.

Next, with the partition set $\Omega : \{\Omega^{(1)}, \Omega^{(2)}, ..., \Omega^{(p)}\}$ derived from the spectrum of L_{rw}, each partition immediately represents a set of correlated experience particles that map to similar fitness values by the property of the covariance matrix *K* in GPR. This serves as a basis for defining the corresponding abstract actions denoted by $A_c : \{A^{(1)}, A^{(2)}, ..., A^{(p)}\}$. Meanwhile, in order to "interpolate" a policy for any given new state-action pair (see Figure 7.3) sufficiently correlated to at least a subset of the older particles, the policy function π is again represented through the fitness function $Q^+(\cdot)$; namely, π is a functional $\pi[Q^+]$ whose probability increases with Q^+ (see (6.9)). Since the value of Q^+ is driven by the experience particles in Ω (see (6.11) and (6.16)), the policy also depends on the particles. With the policy represented through $Q^+(\cdot)$ and the 1-to-1 correspondence between the abstract action $A^{(i)}$ and the partition set $\Omega^{(i)}$, each abstract action effectively encodes an implicit strategy that resolves into a set of primitive actions. Intuitively, if a query state lies in the proximity of those states referenced by a particle set, then applying the actions inferred from these particles will very likely lead to similar results indicated by their fitness values. An example can be observed from the bottom half of Figure 6.7 where each soft state area encloses highly-correlated particles that lead to the same successor soft state. A 3-tier hierarchy for policy similarity can thus be established through the set of abstract actions, each of which reference some set of soft states, which themselves reference a set of correlated particles. Figure 7.2 shows two abstract actions in the bottom row, each of which references a set of correlated particles. Note that in general, the correlated particles need

not reference spatially continuous state/action vectors. This is depicted by the dotted ellipses within the two clusters in Figure 7.2, in which each ellipse (in the same partition) references a set of spatially connected particles (in terms of their referenced states) but nonetheless is apart from those particles in the other ellipses. For instance, the 3 discontinuous regions in Figure 7.3 have a possibility of being grouped into one partition (depending on the choice of kernel function and its hyperparameters) due to their symmetry property reflected also in their correlated fitness values. In particular, different state regions and their associated local polices may be grouped into the same abstract action because their representative particles have similar fitness values. As a consequence, for a given state, in order to resolve a particular primitive action from an abstract action, a matching process needs to be performed. Specifically, this again utilizes the operation of experience association. If the agent currently occupies a state sufficiently correlated to some particles referenced by the abstract action $A^{(i)}$, then this abstract action effectively specifies an *action-selection strategy* that resolves into a set of primitive actions from those correlated particles. That is, those matched particles reference the states consistent with the agent in terms of the kernel-based criterion (i.e. $k(s_h^+, s^+) \geq \tau$ defined in Section 6.7).

Consider a state $s \in S$ in an experience association process with members referenced by an abstract action $A^{(i)} \in A_c$. If $A^{(i)}$ indeed references at least one correlated particle, then $A^{(i)}$ is said to be *in context* with s; otherwise, $A^{(i)}$ is said to be *out of context*. If the chosen abstract action is in context with the agent and also resolves into a primitive action leading up to a relatively higher fitness value, then it is given a relatively higher preference according to the policy representation as an increasing functional; i.e. $\pi[Q^+]$. One caveat arises when the chosen abstract action is out of context, i.e. not referencing any particles sufficiently correlated with the agent's state (i.e. $k(s_h^+, s^+) < \tau$). Since no points of reference exist in such cases, a randomized action-selection strategy is assigned to the abstract action for such states. The implication of a random policy is to be discussed shortly.

Resolving from an abstract action to a primitive action need not be a deterministic operation. However, for simplicity, a greedy policy is used here to resolve from abstract action to a primitive action. By doing so, an experience association will always favor the primitive action from the most correlated particle with a state closest to the query state (e.g. the agent's current state). In the special case where the uncertainty in the action parameters is relatively small, resolving a primitive action from a set of particles would then only require a comparison in the state vector by treating the action parameters as constants; that is, parametric actions in this case are effectively reduced to the action set in standard RL with fixed behaviors. In an experience association process with a larger population of particles, a sample-based strategy can be used in order to avoid an exhaustive search and thus to reduce the computation cost. In particular, the particle reinforcement algorithm introduced earlier in Section 6.7 based on local search can also be reapplied here to effectively reduce the size of the particle set of interest.

7.4 Fitness Value Estimate for Abstract Actions

With both experience association and the concept-driven abstract actions established, each augmented state finally has an intermediate representation from which to derive its fitness value. Recall from Section 6.5 that a fitness value is essentially a utility estimate obtained from the underlying control learner in response to the pairing between a state and an action. As the action is modeled in terms of an operator, the value of its action parameters will only be resolved upon acting on the state (see (6.10) for an example). The resulting action vector combined with the state vector in turn forms an augmented state. The received reward is used to update the utility estimate for this state-action pair from the perspective of the control learner (which is not aware of the underlying action parameters). The only additional step beyond the usual utility update is to relay this utility as the target fitness value for the corresponding augmented state, which together forms a new experience particle represented by the (augmented) vector $(\mathbf{x}_i, q_i)$, and subsequently include it into the existing functional data set Ω.

Thus, a fitness value can be interpreted as an approximate utility estimate that assimilates the uncertainty during the update history for the given state-action pair. In particular, since the augmented state carries extra degrees of freedom from the action parameters, using a utility as a measure for a given augmented state effectively performs a "smoothing" over the variations caused by the stochastic effects inherent in action performance. Similarly, the fitness value estimate for a pair of state and abstract action is effectively a further averaging over the sequence of fitness values from the action-resolution history, which unfolds as sequential pairs of states and parametric actions. Although using the averaged utility as the fitness value is an approximation scheme (compared to a potentially more accurate utility estimate from within the augmented state space), the property of the kernel in addition to the GP prediction allows for a reasonable generalization of value prediction to occur across augmented states. The predicted fitness value will remain accurate as long as some experience particles exist in the neighborhood of the query point. Recall from (6.16) that the mean prediction of the GP is expressed through the kernels centered at all the other augmented states. Thus, the fitness value prediction for any new augmented state is largely determined by those particles with higher correlations. The prediction generalizes further into the unknown parts of the state space through the kernel as a correlation hypothesis. By (6.6), the more neighboring particles exist in the query point, the less the variance of the prediction will be.

The initial fitness value estimate for a given augmented state is the reward discounted by the learning rate (e.g. αr in (4.10)), assuming that all Q-values are initialized to 0. This is similar to the mechanism in utility updates in standard TD learning. However, in order to obtain a cluster-based abstract action set, at least a few experience particles must be gathered. This can be achieved by following a random policy for some period to collect an initial set of experience particles, by which the very first action abstraction model is built. The only way for the "infant agent" to formulate an initial cognitive model of the world is to perform certain trials to gather hands-on experience. Alternatively, it is also possible to formulate the initial action

abstraction with a prior cluster assumption over the similarity among augmented states. Note that using a random policy in such case indeed corresponds to selecting the out-of-context abstraction that carries no relevant context for deducing the policy for a state with no points of reference.

As mentioned earlier, a greedy policy is used to resolve from abstract action to a primitive action by following the kernel-based criterion for associating the related past memory (i.e. particles referenced by the selected abstraction). The subsequent decision making is performed hierarchically with the agent learning the policy over the abstract actions, each of which in turn resolves into a primitive action through the experience association operation. Following the policy over the abstract actions, the agent subsequently executes the primitive action resolved from the selected abstract action (e.g. the one that leads to a highest-valued augmented state). The agent then receives a reward and updates the utility estimate accordingly for the given pairing of state and abstract action, say $(s, A^{(i)})$, by following the update rule of the baseline RL algorithm. With SARSA, this amounts to a generalization over (4.10) to give:

$$Q(s, A^{(i)}) \leftarrow Q(s, A^{(i)}) + \alpha\left[R(s, A^{(i)}) + \gamma Q(s', A^{(i)}_{next}) - Q(s, A^{(i)})\right] \tag{7.7}$$

The utility estimate for the $(s, A^{(i)})$ pair, say q_i, is in turn used as a fitness value for the corresponding augmented state $s^+ = (s, a^{(i)}) = (\mathbf{x}_s, \mathbf{x}_a)$. This produces one complete experience particle: $\omega_i = (s_i^+, q_i)$, where q_i is an estimate for $Q(s, A^{(i)})$.

Because the new data is guaranteed to be correlated to at least a subset of the older particles referenced by $A^{(i)}$, by the definition of experience association, the value $Q(s, A^{(i)})$ carries over to the augmented state $s^+ = (s, a^{(i)})$ as a reasonable fitness value. This can be further justified by the fact that correlated inputs (i.e. $s^+ = \mathbf{x}$) always map to correlated signals

(i.e. $Q^+(s^+)$) by the property of the kernel as a correlation hypothesis, based on which the action abstraction is derived. On the other hand, if the chosen abstraction $A^{(i)}$ is out of context with the agent, the value of $Q(s, A^{(i)})$ cannot be used directly as an estimate for $Q^+(s^+)$ in that $Q(s, A^{(i)})$ only reflects a random choice rather than the utility estimated from a correlated experience. Thus, obtaining an estimate for $Q^+(s^+)$ when $A^{(i)}$ is out of context in principle requires performing experience association against all the known particles in Ω to find the most correlated instance of an experience and its associated fitness value. Again, maintaining a set of state partitions can effectively result in a local search. Details of such Q^+ value adjustments are to be covered in the next section.

Note that the resolution from an abstract action to the corresponding primitive action will gradually tend to a deterministic process as long as the underlying experience particles tend to a stable set, promoting the convergence guarantee of $Q(s, A^{(i)})$. This can be effectively controlled by a gradual increase in the model reformulation period T. As alluded earlier, the control policy π is driven by the experience particles since it is represented through the fitness function Q^+, which is represented by a GP with the covariance function $k(\cdot,\cdot)$. With a softmax exploration strategy similar to (6.9), the policy π over A^+ can thus be represented by:

$$\pi(s, A^{(i)}) = \frac{\exp\left[Q(s, A^{(i)}) / \tau\right]}{\sum_{j \in \{1,..,p\}} \exp\left[Q(s, A^{(j)}) / \tau\right]}$$

$$= \frac{\exp\left[Q^+(s^+ \mid (s, A^{(i)})) / \tau\right]}{\sum_{j \in \{1,..,p\}} \exp\left[Q^+(s^+ \mid (s, A^{(j)})) / \tau\right]}, \tag{7.8}$$

where p is the number of abstract actions and τ denotes a temperature. The term $Q^+(s^+ \mid (s, A^{(i)}))$ in (7.8) denotes the fitness value evaluated at s^+ given the query state $s \in S$

Generalized SARSA($\boldsymbol{k}$,n,T, Ω_0):

// $\boldsymbol{k}$: *The kernel function with hyperparemters* $\boldsymbol{\theta}_0$
// n: *Number of action abstractions*
// T: *Model reformulation period* (i.e. GPSC)
// Ω_0 :*initial source of experience samples*

1. Formulate an initial model $A_{c0} : \{A_0^{(i)} \mid i = 1 \sim n\}$ using an initial sample set Ω_0 via **GPSC** procedure (see section 7.2)
2. Initialize $Q(s, A^{(i)})$ arbitrarily.

Repeat (for each episode)

3. Initialize s. //*starting state*
4. Choose $A^{(i)} \in A_c$ from s using $\pi(s, A^{(i)}) = \pi[Q^+]$ (e.g. softmax in (7.8)) where $A^{(i)}$ resolves into a primitive action $a^{(i)}$ via **experience association** with $(s, A^{(i)})$.

Repeat (for each step of episode)

5. Reformulate A_c using **GPSC** every T steps.
   5.1 (Optional) $T \leftarrow T + f(T)$ where f maps T to some number as an increment over T. // *gradually stabilize cluster structure*
6. **Take action** $a^{(i)}$, observe r and s'
7. Choose $A^{(j)} \in A_c$ from s' using $\pi[Q^+]$ where $A^{(j)}$ resolves to $a^{(j)}$ via **experience association** over $(s', A^{(j)})$. // *logically identical to step 4*
8. Update $Q(s, A^{(i)})$ using the update rule in (7.7).
9. $s \leftarrow s'$, $a^{(i)} \leftarrow a^{(j)}$ // A_c *depends on step 5*

Until s is terminal

Figure 7.4 G-SARSA.

performing an experience association operation with $A^{(i)}$. The value of $Q^+(s^+ \mid (s, A^{(i)}))$ is estimated through the GP prediction based on the particles in the memory (i.e. Ω). A generalized SARSA (G-SARSA) is illustrated in Figure 7.4.

7.5 Particle Promotion and Particle Updates

Recall that the utility estimate obtained via evaluating the policy over abstract actions is carried over to serve as the fitness value for the corresponding augmented state. This is only reasonable under the assumption that the chosen abstract action is in context with the agent

(specifically, the current state of the agent). Only under an in-context scenario will the resulting utility estimate be consistently correlated to those of the other experience particles. This is due to the fact that each abstract action is formulated by grouping correlated particles and thus can only be used to match a correlated hypothetical state (Definition 6.5). In contrast, the randomized action-selection strategy in the out-of-context cases will result in utility estimates reflecting only random choices of actions, which without loss of generality leads to relatively lower utility values. In particular, for the case of a random policy, the hypothetical state is formulated by merging the value of the queried state with the action vector resolved from a random action choice, which corresponds to a pure exploratory strategy. Note that the random policy also applies to an abstract action not referencing any particles (i.e. an empty cluster), which automatically implies the out-of-context scenario. While the out-of-context abstract action is in general not preferred in the presence of other, better choices, nonetheless the random policy serves as a necessary exploratory mechanism for the agent to discover good actions in the unknown decision contexts. Yet, in this case, the resulting experience particle in general does not truly reflect the selected abstract action and hence its utility estimate cannot be directly used as the fitness value. One way to infer the fitness value for such out-of-context particles is to apply the experience association against a selective set of other particles in Ω to find the most correlated match and subsequently use its fitness value as the final estimate. This process is addressed as the particle promotion operation to indicate the fact that the particle's fitness value, originally biased toward a random choice, is being "promoted" to match those of the in-context experience particles. By this token, the particle can be incorporated into memory for estimating the policy over its neighboring augmented states. To reduce the computational cost of the particle search, it is helpful to maintain a set of state partitions such that each partition references its local set of particles. In this manner, the particle promotion only requires a local search and consequently its operation effectively coincides with the particle reinforcement algorithm presented in Figure 6.4. If, however, the new particle does not match with any existing

particles (i.e. $k(s_h^+, s^+) < \tau$ for all possible s_h^+), then its utility estimate obtained from the control learner is used directly as the fitness value. In such cases, the particle corresponds to a new experience with no points of references and thus is logically identical to the particles formed at the initialization stage during which the utility estimate amounts to a discounted reward (by the learning rate).

Alternatively, particle promotion can also be realized using the GP prediction directly. However, unlike the policy representation through the GP prediction (i.e. $\pi[Q^+]$), errors in the fitness value estimate can result in skewed functional data, which will inevitably propagate the error further to the future prediction. In particular, the precision of $Q^+(\cdot)$ as a GP depends on the correlation between the query point and all the memory kept in Ω. Thus, the error propagation can occur when the agent tries to predict the value of an out-of-context particle with its state spatially too far away from those in the memory. Thus by (6.6), the fitness value estimate can assume a wide range of possible values. To remedy the impact from the error, one would need to introduce a variance-specific threshold in order to filter out the particles with unreliable fitness values. However, setting an appropriate threshold is not straightforward due to its dependence on the entire data set. For reasons above, applying experience association to the existing particles is a preferable choice for the implementation.

7.6 Empirical Study

To motivate a more complex parameterization of actions as a setup for formulating the concept-driven abstraction, a task assignment domain is presented in this section. As a review to the task assignment scenario motivated by PanDA-PF WMS and introduced in Chapter 2, consider the case where a continuous stream of user tasks is to be assigned to a resource pool (e.g. a pilot-distributed Grid network) with a number of servers having time-varying resource capacity (e.g. load average). In particular, the real-time resource profile varies as a result of the sharing from multiple users. The primary goal for an ideal scheduling policy is to match the

servers with compatible user tasks such that a given performance metric is optimized (e.g. minimum turnaround time, etc). In this scenario, simply enumerating possible assignments to servers using a standard RL formulation would fail to reach a stable policy due to the time-varying properties of the target servers. Multiagent formulations under standard RL often introduce combinatorial state and action spaces by considering each task and server as an intelligent agent with its own state and action space. This often leads to a combined state space with a large dimension, making it difficult to compute utilities efficiently with a higher population of agents. On the other hand, with parametric actions, each task assignment as a control decision is modeled by the dynamically changing yet shared attributes associated with the target servers. For example, the percentage CPU time, remaining memory, etc, are among the candidate action parameters since it is these attributes that determine the server performance. While the primitive action set in this case still corresponds to all the available servers (which may also vary over time), all the actions as assignment decisions effectively share the same parameters and hence are not subject to the combinatorial explosion due to the increase of servers. In addition, a varying action set will not impact the learned policy for task assignments since the action parameters effectively control the policy rather than the action set per se. Similarly, the state space represents task properties (e.g. job types, CPU demand, etc) in addition to perhaps some global statistics over the resource pool. In this formulation, the challenge is to identify the functional relation inherent in the state features and action parameters in addition to their degree of relevance towards the final prediction for any given task assignment.

To demonstrate G-SARSA using the CDLA framework introduced in this chapter, a task scheduling simulator has been designed that simulates a continuous stream of user tasks and candidate servers of varying resource capacity. The specification of the experiment is given in Figure 7.5. The goal is to determine an assignment policy that matches user tasks with compatible servers in order to optimize a given performance metric (e.g. minimum turnaround

Parameter Spec.	Values
Task feature set (state)	***Task type**, size, expected runtime*
Server feature set (action)	***Service type**, percentage CPU time, memory, disk space, CPU speed, job slots*
Kernel *k*	*SE kernel + noise (see (6.3))*
Num. of Abstract Actions	*10 (assumed to be known)*
Model update cycle T	*10 state transitions towards 100*

Figure 7.5 Specification on task scheduling domain.

time). In this study, a match between a task and a server requires the *task type* to be identical to the *service type* of the server (see Figure 7.5). This is to simulate the condition of the match-making process used in the Condor system [55-61] in which the requirement statements of the task and the server must agree before a match can occur. An example of a realistic requirement statement for a user task can involve the demand for a minimum CPU speed, memory size, or constraints on the platform as a computational environment. Similarly, a server can also have a requirement defined that only accepts certain job types or users. Thus, a task will only execute on a server when both ends have compatible requirement definitions. For simplicity, the matching condition is simulated by checking the consistency between *task type* and *service type*, each of which is part of the features that model a task and a server, respectively. If the matching condition holds, then the turnaround time is minimized when the target server has a high availability and relatively higher processing power, leading to a higher reward. Conversely, a task will fail to run under the condition of a mismatched requirement or insufficient resource capacity, all resulting in negative reward. Note that a matching requirement statement still does not guarantee success in terms of the task processing result. A deficient resource capacity relative to the task demand can also lead to a failed task, resulting in a negative reward identical to the mismatch case. This occurs, for instance, when the given task requires more memory

than what is currently available in the matching resource. Similar to the navigation domain presented in Section 6.8, each feature characterizing the task and the server are modeled by constrained random variables following a designated probability distribution. In this study, the yield in the form of (6.1) with a Gaussian random variation superimposed on the target value is used to simulate both the state and action features.

However, to focus on the MDP approximation scheme, the state features (which model incoming tasks) are assumed to be static and fully observable once their values are determined (during the initialization phase in the simulation). That is, the task-specific feature values do not change over time and are fully-accessible without added noise. In contrast, most of the server-specific features (i.e. action parameters) continue to vary after the assignment of their initial values with an exception of one feature: *service type*. Since *service type* is used to simulate the requirement statement enforced by the machine owner, we assume that the statement does not vary over time as opposed to the other features that characterize dynamically-changing resource profiles. Doing so effectively simulates the condition that the resource capacity varies over time due to the sharing of multiple users across the network in addition to the varying distribution of incoming tasks. Moreover, the availability of the servers upon selection (i.e. the decision of a task assignment action) also varies according to a designated probability distribution. This is to simulate the condition where the server during the match-matching phase may not be always available due to limited job slots (as they are being occupied by other incoming tasks). Also, recall from the context of PanDA-PF WMS, the pilots will automatically exit if no matching tasks are found in the PanDA server or if they somehow failed to establish communications (in terms of periodic messages illustrated in Section 2.3.1). Under these circumstances, the pilot-resident servers, albeit known to the RL agent, may not be always available as task-assignment action choices.

With the standard RL formulation, the action set is determined by enumerating all the possible server allocations, which in general does not enable the agent to adapt the learned

policy to the time-varying machine profile that characterizes the servers. Consequently, standard SARSA is not an ideal policy learning algorithm for this domain. The key issue is that the same task assignment (by holding the task and server fixed) could have varying performance over time depending on the server availability and the remaining resource. Thus, the optimal policy identified earlier may not carry over to the future. In CDLA, on the other hand, the real-time server property is characterized by the action parameter, reflecting the task assignment action as a dynamic process contingent upon the various server attributes such as the CPU percentage, time, etc. In this experiment, 6 action parameters are chosen to model a server, including CPU percentage, remaining memory, etc (see Figure 7.5). In particular, user tasks are represented by 3 features modeling the state, including the matching criterion (i.e. *task type*) and 2 correlated features: task size and expected runtime. Under these settings, each decision produces a 9 dimensional augmented state (with 3 task-specific state features paired with 6 server-specific action features). With the correlation assumption in the form of the kernel in (6.3) (i.e. the SE kernel with weights associated with each feature dimension), each coordinate would contribute to the fitness value prediction differently. The task is thus to identify their relevance degree to the fitness value for a given state and action combination (as an assignment decision). For instance, identifying an abstract action representing a "matching concept" requires the agent to identify *task type* and *server type* being equal for a higher fitness value to occur. Figure 7.6 shows the experimental result comparing generalized SARSA (G-SARSA) against a random policy and a stripped-down version of the Condor's match-making algorithm, which always ensures a match prior to the task assignment. Figure 7.7 shows an example of the abstract actions learned during the policy learning process. For instance, an abstract action that represents a *matching concept* is identified by the augmented state in which the state feature *task type* is identical to the action parameter *service type* (the top 3 rows highlighted in blue). In contrast, non-match abstractions at the bottom two rows (in yellow) lead to relatively lower fitness values and thus are not preferable if a matching abstraction exists.

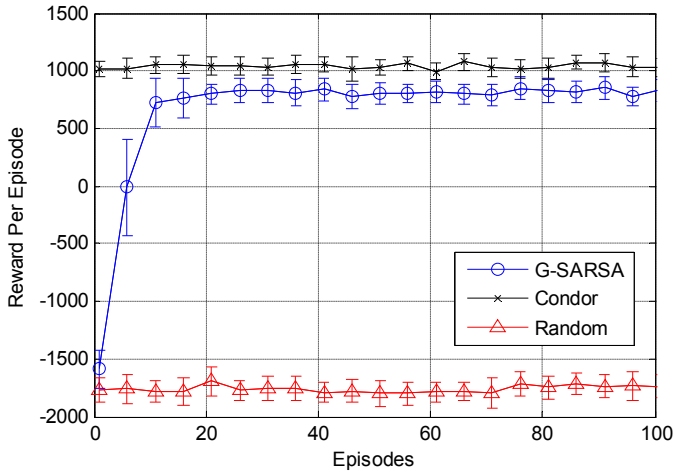


Figure 7.6 Performance comparisons with G-SARSA, a Condor-flavored match-making algorithm and a random policy. The softmax exploratory strategy used by G-SARSA is eventually reduced to the ε-greedy strategy at low temperature.

	Task type	Size	Expected runtime	Service type	%CPU time	Fitness Value
1	1	1.1	0.93	1	9.784	120.41
2	2	2.5	1.98	2	10.235	128.13
3	3	3.2	2.92	3	15.29	135.23
4	1	1.0	1.02	2	20.36	-50.05
5	2	2.0	2.09	3	0.58	-47.28

Figure 7.7 Illustration of 5 different learned decision concepts. The top 3 rows in blue indicate success matches while the bottom 2 rows in yellow indicate failed matches. E.g. for tasks of type 1, the ideal policy is the assignment to the servers of service type 1 by selecting abstract action 1, leading to higher fitness values.

Consequently, the most preferable strategy is for a task to always select an abstraction with matching servers and, better yet, with the associated servers having a relatively higher resource capacity with high probability.

Note that in contrast to the navigation domain in Section 6.8, the action operator is only defined implicitly in this domain. Recall that an action operator takes on an input state, resolves

the stochastic effect in the action and then finally produces the next state according to the rule defined in terms of the form of the operator. Since the action in the navigation domain has a direct relation to the next state (which depends on the current position and the direction along which the agent chooses to move), it is straightforward to specify the operator in the form of a matrix (e.g. (6.10)). Similarly, in a linear dynamic system, for instance, the next state can often be expressed in terms of matrix multiplication against an input state. However, in the real-world task assignment domain, the incoming task as the current state does not necessarily have an identifiable relation to the next task (as the successor state) in a manner that depends on the chosen server (corresponding to a task-assignment action). For one thing, a candidate server is often shared by multiple users in the context of PanDA-PF WMS as well as other common scenarios in the Grid. Also by assumption, the user does not have prior knowledge about the server set available to his or her tasks. As a result, the task stream is often independent of the target server although it is possible for any two tasks to assume interdependence. For simplicity, this study assumes that the successor task is independent of the server to which the current task is assigned. As a result, it is not necessary to explicitly specify the action operator that relates the current task to the next apart from the modeling of the action parameters collectively as a random process. Nonetheless, the notion of action operator still applies in this study since the augmented state, as the byproduct of an action operator, is produced in the same manner as in the navigation domain through the following logical sequence: i) an input state is made available as an incoming task awaiting for a matching server ii) the implicit action operator takes on this input task, resolves the stochastic behavior of the action by taking a snapshot of the resource capacity of the chosen server upon the assignment of the task iii) the next task again is made available as the next state, although this time, the new task does not have a direct relationship with the predecessor task in terms of the parameter characterizing the server.

7.7 Summary and Future Perspectives

This chapter developed a generalized RL framework, CDLA, from the theoretical perspectives of kernel functions used as a correlation hypothesis that connects the policy-embedded particles across the augmented state space into a set of coherent decision concepts. In particular, the very same kernel is shared by both the Gaussian process as a fitness value predictor and by the spectral clustering procedure that formulates the abstract action model. The kernel as a covariance function in GP effectively associates similar augmented states in terms of the correlation degree of their fitness values. In other words, the more correlated the augmented states are, the closer their outputs are as fitness values. Through this property, the covariance matrix can effectively be reused as an input similarity graph for the spectral clustering agent to discover homogeneous patterns of augmented states with similar fitness values. Each pattern is used as a representation for the abstract actions, out of which the RL agent derives a control policy. Subsequently, by the RL agent performing experience association against each abstract action, one (or more) correlated particles are identified. The selected particle holds a primitive action (with particular parameter values) applied earlier in a state similar to the current state of the agent. Therefore, as long as the fitness value of the particle is relatively higher, this action is again preferable for the current state. The objective of CDLA is to provide a compact representation with policy generalization capacity through identifying conceptually-coherent decisions made in the past. Similar decisions form a policy-guiding concept, thereby enabling the RL agent to have a basis to “understand” what the decision process entails and thus becomes more adaptive to the varying environment.

In the section of empirical study, we have demonstrated that the abstraction formulated under CDLA uncovers the correlation between the state features and the action parameters. When applied to the task assignment domain, such correlation pattern can be used as a pointer for specifying the requirement statement necessary to promote better match between user tasks and computational resources.

On the other hand, the CDLA framework developed in this chapter can also apply to POMDP domains where states are not fully accessible to the agent. The key lies in the fact that the role of kernel functions as a correlation hypothesis has a "symmetry property" over the pairing of state and action vectors (also see Figure 6.7). Although actions are modeled in terms of constrained random processes and do not always resolve to the same action vectors (see Figure 7.3), the generalization property of the kernel enables value predictions to occur in the neighboring point of the augmented state. That is, as long as the augmented state resolved from the merging of an active state and an action (that fluctuates) is sufficiently correlated to other experience particles held in memory, the predictive fitness value will still be upper bounded by a small variance (see (6.6)). This is essentially the key essence of the soft state (see Section 6.7.2), which enables the state transition to become "fuzzy" as opposed to the concrete state transition in standard MDP. The stochastic action effect can in principle be translated to the corresponding state such that the net result of the state transition ends up being equivalent (upper row of Figure 6.7). As a result, the generalization property of the kernel mentioned above can again be applied to stochastic states such as the belief state formulation. The connection between the notion of soft state and the belief state model is provided in Section 6.7.

In addition, a probabilistic model for resolving the primitive action from the abstraction is also possible by exploiting the property of the Markov transition matrix inherent in the random walk graph Laplacian for better policy adaptability. For instance, by using the Nadaraya-Watson estimator introduced in Section 6.7, the experience association can be realized probabilistically with a higher probability given to those actions resolved from highly-correlated particles that represent more closely-related past memory.

The generalized reinforcement learning framework in this dissertation has been developed with the consideration of the applicability to general task domains in addition to the context of a generic workload management system. CDLA provides a flexible framework in that

the entire policy learning mechanism is kernelized (also see Section 5.4), meaning that the kernel function as a state correlation hypothesis can be extracted as a separate layer for the learning process. The kernel-based representation penetrates throughout the entire learning architecture including the representation for the reinforcement field, the operation of experience association, value function approximation, policy function learning and the abstract concept formation (see also Figure 7.1). By changing the form of the kernel function, the agent can effectively alter the mechanism of associating the past memory, leading to a different concept formulation over the past learning experiences. This is effectively a prototypical mechanism in altering the cognition of the learning agent in terms of the way it interprets the past, which in turn adjusts the ongoing control policy accordingly. Nonetheless, under any choices of kernel functions, the policy learning is still driven towards the global objective – maximizing the accumulated reward – by virtue of particle reinforcement (see Figure 6.4). Through the particle reinforcement mechanism, the agent will preserve those instances of memory that tip the scale towards increasing the utility; meanwhile, the negative experiences are also retained in memory as counterexamples so that the same mistake can be avoided in the future. Through this, the theoretical foundation in GPR and functional analysis can be brought to bear upon the generalized RL framework in terms of its robustness, scalability and extensibility.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusions: Putting Things Together

In the initial phase of this research, we have developed a generic workload management system, PanDA-PF WMS, with decentralized components for task submission, scheduling, data movements, resource discovery and the final task dispatch to the computational resource of choice. In particular, the pilot factory mechanism enables a scalable pilot dissemination to candidate computational resources such that resource discovery and task assignments can simultaneously occur as the end user continues to submit their tasks. From the architectural design perspective, the PanDA-PF WMS maximizes the resource usage in that any available candidate machines appear on-demand for the incoming task stream due to the parallelism between distributed pilots in action and the continuous task dispatch from the queue (in the PanDA server). In addition to the use in standard Grid settings, the flexible architectural design in the PanDA-PF framework can potentially adapt to multiple forms of computational resources in a general resource-sharing environment, which would include the volunteering computing system (e.g. PCs connected via the Internet), and dynamic virtual clusters (e.g. DVC, Condor glidein pool, etc) among others through pilots serving as client agents requesting compatible tasks.

In order to fully utilize the computational power opportunistically from within the constantly varying resource profile, it is imperative for the task assignment policy to become adaptive in a manner that stays consistent with the true, real-time system dynamics as much as possible. To achieve adaptive policy learning with the consideration of general applicability in complex control tasks, the second part of this research concludes with a generalized

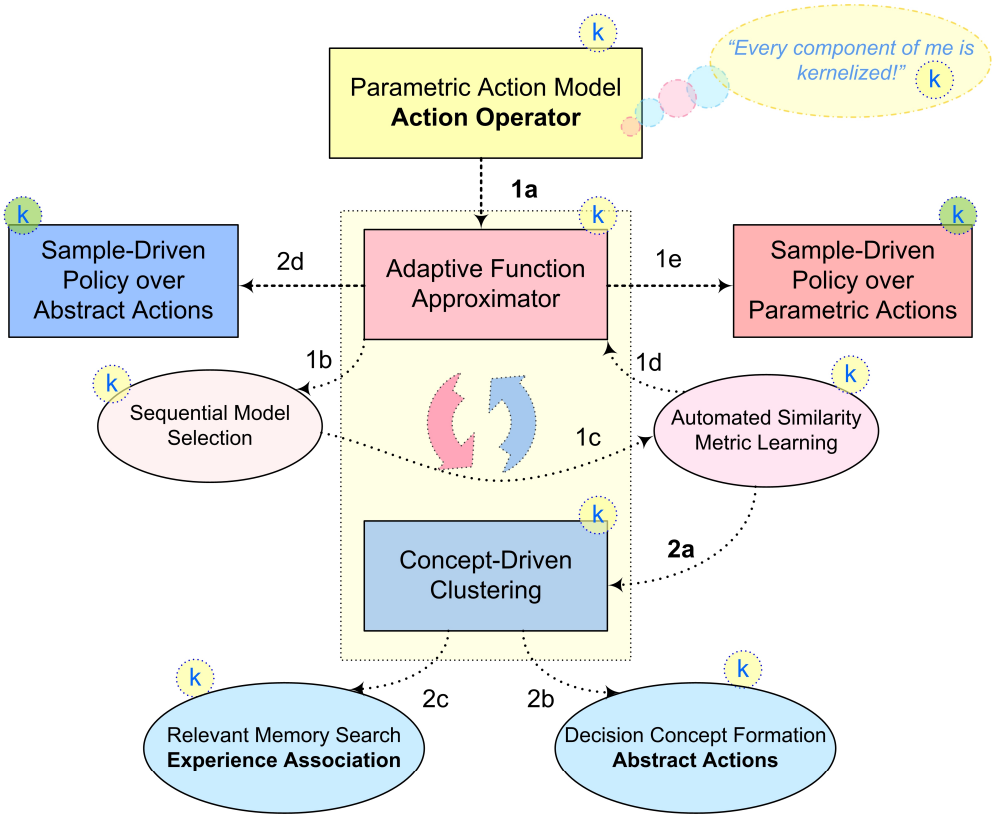


Figure 8.1 CDLA schematic. Periodic iterations between value function approximation and concept-driven cluster formation constitutes the heartbeat of CDLA. The conceptual sequence from 1a to 1e corresponds to the logical flow that progressively updates the reinforcement field as a policy generalization mechanism. The sequence from 2a to 2d on the other hand references the logical flow towards establishing an abstract control policy based on a set of coherent decision concepts as high-level control decisions.

reinforcement learning framework, CDLA, by integrating the reinforcement field and decision concept formation into a coherent learning process represented by the center box in Figure 8.1. In particular, the novel learning architecture addresses the core issue of standard reinforcement learning algorithms from the perspective of parametric action formulation, extended state space representation and sample-driven policy representation. All the components in CDLA are represented directly or indirectly through the kernel function that serves as the fundamental building block for value prediction, concept formation as well as policy search. In Figure 8.1, kernelized components are marked by a circled *k* in yellow whereas indirectly-kernelized

components are marked by their green counterpart. Both of the policy functions (over parametric actions or over abstract actions) are indirectly kernelized due to their dependency on the kernelized functional approximator.

From the observation in a realistic task-assignment domain introduced by PanDA-PF WMS, the pilot-resident machines can in principle become so large that deriving an effective task-assignment policy may be infeasible. This is due to the fact that available task-assignment actions will become large as pilots are disseminated to more candidate machines. Moreover, the task assignment targets can also vary over time as pilots join and exit the target hosts. In Chapter 4, we arrived at our first action abstraction by clustering similar actions with the incentive of reducing decision points per state. In other words, characterizing a pilot-resident host with its dynamic attributes (e.g. CPU percentage, memory, etc) and modeling each action as a possible task-assignment to a candidate host allows us to obtain an abstraction that aggregates similar hosts together. In this manner, by applying CAQ-learning developed in Section 4.4, the eventual task-assignment policy does not have to depend on individual hosts as control decisions but instead only on the "clustered actions," each of which references a set of relatively homogeneous machines, performance-wise, with similar dynamic properties. However, a strong assumption was made in order to simplify the treatment of the action abstraction; that is, the correlation between action parameters and state features was assumed to be a prior knowledge in addition to their mappings to reinforcement signals. In this way, the cluster assumption over similar actions can be considered as a heuristic for clustering actions. However, in most complex domains of interest, deducing or implementing such a cluster assumption is often not straightforward.

Next in Chapter 5, we drew the connection between similarity measure and regression in order to set the stage for obtaining more general and perhaps, more compact representations for approximating the potential value function used in reinforcement learning algorithms. In this manner, it becomes possible to integrate the notion of a similarity measure (also as a cluster

assumption) to the value function approximator such that value predictions and clustering similar state-action pairs can be performed simultaneously. An added advantage that almost comes without cost is that we no longer need to confine the cluster structure within the action parameter space but instead expands into the state space so as to express the notion of context-dependent control policy. In particular, from a simple linear model, ridge regression to Bayesian linear regression, we see that a function approximator can often be expressed as a linear combination over the training data. By using a selected basis set (see Section 5.2) of sufficient size, one can in principle express arbitrary non-linear curves. Perhaps a more important observation is that the predictive function of interest lies in the inner products of training samples and their linear combinations. By incorporating the notion of kernel functions (see Section 5.3), one can thus represent the predictive function in terms of a linear combination of kernels, in which the kernel can be used also as a cluster assumption over states. This is due to the fact that a positive-definite kernel effectively corresponds to an inner product in the Hilbert space. Interested readers are also encouraged to refer to [1, 22] for more examples and justifications. With the foundation established so far, we then proceeded to integrate the parametric-action model, state similarity measure, value function approximator, and cluster-based abstraction formation as a coherent learning framework, in which all of the logical pieces can be kernelized.

In particular, this dissertation generalizes standard RL through a two-phase construction shown in Figure 8.1 in which each phase is represented by a conceptual sequence illustrating its logic flows. The first phase (1a-1e in Figure 8.1) introduces the notion of the reinforcement field that serves as a policy generalization mechanism as well as a compact, kernel-based representation for approximating a value function that operates over the extended action formulation – action operator. The second phase (2a-2d in Figure 8.1) builds on top of the reinforcement field and subsequently formulates the action-oriented policy abstraction to reduce decision points per state by making use of the trained kernel as a cluster assumption.

This is depicted via the logical flow from 1c to 2a in Figure 8.1. Common patterns associated with each cluster-based abstraction can thus be used to identify the conditions that determine compatible bindings between the state and the action toward a higher long-term payoff. This effectively defines a context-dependent control policy such as the match-making criterion for compatible tasks and servers illustrated in Section 7.6. The interleaving process between value function approximator and the policy-inducing concept formulation thus leads to the concept-driven learning architecture (CDLA) that connects all kernelized components together to derive abstract control policies.

8.1.1 RL Generalization I: Introducing Action Operator and Reinforcement Field

The first stage of RL generalization includes a more flexible action expression – action operator – that connects the local dynamics induced by the action with its associated state transition. The action operator defines the state transition operationally such that the action can be considered as the property of an "external field" that acts on an input state and subsequently yields another state following the rule defined by the field. In this manner, the policy search can be integrated as a unified continuous process with the state transition (as a sequence of local constraints) that leads to a maximized accumulated reward as the global objective (which originates from an analogy to the Lagragian mechanics in Section 6.6). We will have more to say in the next section about such a formulation. To specify an action operator, first we parameterize the action in terms of a random vector in which each coordinate follows a task-dependent probability distribution with a bounded support. The state transition can thus be specified in the form of a matrix multiplication where the matrix (often symmetric) acts as the operator that takes an input state vector and subsequently generates an output as the successor state (see Section 6.6). This in turn facilitates pattern discovery through identifying commonality in action dynamics and their influence over the state space.

An augmented state is produced as the byproduct of the action operator acting on an input state. The augmented state essentially pairs a state vector with a (constant) action vector resolved from the stochastic action effects; that is, an implicit sampling process is performed along each coordinate of the action modeled by constrained random variables (see Section 6.2). The kernel as a state correlation hypothesis then takes the new augmented state as an input such that the kernelized augmented state merges with the other policy-embedded kernels to form an updated reinforcement field (see Section 6.6). The representation of the reinforcement field is here a direct consequence of using GPR as a kernel machine such that the value prediction assumes the form of a linear combination of kernels, each of which is "centered" at an augmented state. Notice that the kernel centered with a pattern, i.e. $k(\cdot, \mathbf{x}_i) = \Phi(\mathbf{x}_i)(\cdot)$, is again a function that expects another pattern (i.e. augmented state in our case) as an input. Thus, a linear combination of these policy-embedded kernels collectively forms a functional vector field – a continuous policy-generating field so to speak.

By the property of GP-based value prediction, any augmented states that are sufficiently correlated to the states already embedded in a kernel expression will also have a correlated predictive value. That is, a high correlation within the input pairs also correspond to correlated scalar outputs. For this reason, an augmented state essentially represents a localized control policy that also applies to the neighboring (augmented) states through the generalization property of the kernel (also see Section 6.7.2). As a consequence, by representing the policy as a functional of the GP-based function approximator (see Section 6.5), the policy can thus be reasonably estimated in the unexplored state space as long as there exist neighboring evidence represented by other policy-embedded kernels. Note that the GP-based value function approximator can be periodically updated and optimized through maximizing marginal likelihood with respect to the kernel hyperparameters via the ARD procedure (i.e. sequential model selection in Figure 8.1). In this manner, the control policy is progressively updated (1c-1e in Figure 8.1) such that it reflects the current dynamics of a

potentially non-stationary environment. Also, the control policy can be expressed in terms of a field plot in which each position represents a state with a corresponding arrow pointing in the direction consistent with the control policy. If a state area is sufficiently explored such that an optimal policy is identified through the evidences gathered within the area (i.e. experience particles), then the arrow will in principle point in the direction that fulfills the global objective (i.e. maximum accumulated reward). Conceptually, the reinforcement field plot is not restricted to 1-D to 3-D settings such as those illustrated in Section 6.8. In a task-assignment domain, for instance, since the state and action vector can both live in a higher dimensional space, it becomes infeasible to actually plot the field.

The formulation of reinforcement field is indeed an application of using kernel functions as a (state) similarity measure, which is often a constituent part of any linear smoother as a function approximator (see Section 5.1). Since a similarity measure is effectively used as a cluster assumption for clustering algorithms, the formulation of the reinforcement field also leads to a natural representation for deriving cluster-based abstractions (i.e. 1c to 2a in Figure 8.1).

8.1.2 RL Generalization II: Concept-Driven Abstraction

Recall that in our first action abstraction in CAQ-learning, we attempted to cluster similar actions in order to reduce decision points per state. This is helpful especially in the domains involving an irreducible yet large set of actions. In order to develop a match-making policy for input tasks against a large set of computational resources, it may not be strictly applicable to use methods based on Sequential Monte Carlo (SMC) sampling [113]. In the SMC-based approach, *importance sampling* [107] is applied over the action space (often continuous) to extract “dominating actions” under which the resulting control policy will lead to relatively higher utility. However, the resource allocation strategy (or equivalently, task assignment strategy) in the context of PanDA-PF WMS requires the policy to include all pilot-resident hosts (potentially large set) as possible candidates. On the other hand, it may be

tempting to discretize the parametric action model such that the multivariate representation of actions can be decomposed into a finite set of actions (e.g. as a preprocessing step for the sampling-based approach); that is, by varying each coordinate in the unit of a predefined interval leads to a set of discretized actions. However, the resulting action set is often either significantly large or not meaningful. Doing so also loses the ability to adapt to possible action variations addressed by parametric actions (and its derivative, the action operator). The key lies in the fact that action parameters often need to be simultaneously controlled in order to represent a meaningful action in many domains. In particular, consider a candidate host that is characterized by a set of parameters representing the related resource capacity (e.g. CPU percentage, disk space, etc). These parameters are often interrelated and need to be determined together in order to represent the resource profile associated with a host. This is also true, for instance, in the control domain of a robot arm in which one may wish to find the best trajectory from a start to the goal position in order to fetch a particular object or avoid certain obstacles. A particular movement, depending on the parameterization scheme (e.g. using position, velocity, and acceleration variables to express joint torques [14, 114]), requires a simultaneous control of the parameter set. That is, a sensible and effective arm movement may not be expressed simply in terms of a sequential increment (or decrement) from one parameter to the other, each corresponding to a discretized action choice.

For reasons above, we took the route of formulating a cluster-based action abstraction with the following advantages: i) all the available actions are preserved ii) the notion of action operator still applies and iii) a reduction of a large action population to a smaller coherent set of high-level abstractions can be achieved that also take into account their associated states due to the augmented state representation. In this manner, sampling-based methods can still apply in resolving a particular primitive action choice from its affiliated cluster but from within a potentially much smaller population. Better yet, the property of the kernel allows us to implement the notion of experience association (see Section 6.7). The operation of experience

association effectively searches from a relevant piece of memory (i.e. correlated evidences grouped in a cluster) and subsequently extracts the best instance of experience, referencing a preferred action that leads to a higher utility. In this manner, each high-level abstraction, when in context with the current state of the agent (Section 7.4), can effectively represent an action-selection strategy. Conversely, if the selected abstraction is out of context, meaning that the referenced evidences are not correlated to the current state by virtue of experience association, then a random (primitive) action choice is selected. Such a randomized action also in a way represents a "creative decision" through the process of particle promotion (see Section 7.5). It is said to be creative due to the following two reasons that are interrelated: i) randomly picking an action serves as a necessary exploration strategy to identify good actions in unknown parts of the state space; yet with the particle promotion as a supporting mechanism, a random guess is no longer secondary to the policy being followed especially when the agent is required to make a decision where there is no point of reference in memory ii) if the resulting augmented state associated with this random action (as a new evidence) happens to match (again by virtue of experience association) with one or more old evidences in the memory, then the (fitness) value of this new piece can be estimated with a relatively low variance by the property of the (functional) cluster-based abstraction. By ii), this out-of-context evidence can still be incorporated into the memory (as a useful piece of information) that will in turn help with the following cycles of value predictions and policy evaluations. In particular, since the evidences in the memory are represented in terms of a functional dataset (Section 6.3), any pair of correlated evidences thus also share correlated reinforcement signals (i.e. fitness values in Definition 6.2) through the property of the covariance matrix used in GPR. A fully-connected similarity graph can be constructed directly by using kernelized entries in the covariance matrix as weights. In this manner, a higher weight corresponds to a highly-correlated (augmented) state-action pair with correlated fitness values. The graph Laplacian derived from the similarity graph is subsequently used as an input for a spectral clustering mechanism to formulate a coherent set

of policy abstraction used as control decisions (Section 7.3). Each cluster derived in this manner is a functional cluster (see Section 7.2) since correlated inputs (augmented states) are grouped into the same cluster in accordance with their correlated fitness values as functional responses. If the graph Laplacian embeds a Markov transition matrix, each cluster can be interpreted as referencing a set of (generalized) state-action pairs that tend to adhere to each other with high probability with respect to their corresponding (soft) state transitions (see also Section 6.7.2).

In addition, with a properly parameterized kernel function, the cluster can also be shown to reveal the correlation structure across each feature dimension. In particular, we used an SE kernel with each feature dimension weighted by a hyperparameter as an example (see (6.3)). Subsequently, through the optimization process via ARD, the optimal configuration of hyperparameters can then be used to identify the condition to hold between the state features and action parameters such that a cluster, when chosen as a high-level control decision, corresponds to a benign concept (with its cluster members referencing relatively higher fitness values). Similarly, one can also identify malicious concepts represented by the clusters with members referencing relatively lower fitness values. In this way, each cluster effectively represents a decision concept. To this end, we showed in a simulated task-assignment domain mimicking the environment induced by PanDA-PF WMS that CDLA can effectively identify the requirement statement necessary to optimize the final task-assignment policy.

8.2 Future Work

Future work will involve further assessments of the RL framework developed in this research in various application domains and its extension in POMDPs and with probabilistic experience association as mentioned earlier. In particular, since every logical piece can be kernelized, learning an effective functional form for the kernel can potentially improve the performance of CDLA on multiple levels through the interconnected learning processes that encompass value function approximation, policy search, experience association, and abstract

concept formation. Note that one can also construct new complex kernels out of simple kernel functions as building blocks. Note also that the input to the kernel are certainly not restricted to vectors but can also include more complex or even symbolic objects including sets, graphs, strings [116], tensors (effectively a generalization over vectors) [115], etc, as long as they correspond to valid inner products in the feature space. Thus, it is of interest to find other richer expressions for the pairing of states and actions representing a localized policy. In addition, the notion of action operator can in principle extend to different types of operators [104], including differential operators used extensively in linear dynamic systems that relate the current state and the next state in terms of time derivatives. In such cases, diffusion kernels [112] and other graph kernels derived from the graph Laplacian, for instance, can be used to express the degree of correlation over time-dependent states. Subsequently, the chosen kernel can be used as the correlation hypothesis to establish the reinforcement field from which to derive the control policy over the kernel-induced action abstraction within the system of interest.

REFERENCES

[1] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge Univ. Press, 2004.

[2] Konidaris, Osentoski, "Value function approximation in reinforcement learning using the Fourier basis," Technical Report, UMCS-2008-19, Dept of Computer Science, Univ. of Massachusetts, Amherst.

[3] S. Mahadevan and M. Maggioni, "Value function approximation with Diffusion Wavelets and Laplacian Eigenfunctions," In Proceedings of the Neural Information Processing Systems (NIPS). MIT Press, 2006.

[4] M. Hauskrecht, "Value-function approximations for partially observable Markov decision processes," J. Artificial Intelligence Research, vol. 13, pp. 33–94, 2000.

[5] S. Mahadevan, "Proto-Value Functions: Developmental Reinforcement Learning," in Proc. of the Int'l Conf. on Machine Learning, 553–560.

[6] S. Mahadevan, M. Maggioni, K. Ferguson, and S. Osentoski. Learning representation and control in continuous markov decision processes. In Proceedings of the National Conference on Artificial Intelligence (AAAI), 2006.

[7] D. Ormoneit, S. Sen, "Kernel-Based Reinforcement Learning", TR 1999-8, Statistics, Standford, 1999.

[8] R. Sutton and A. G. Barto, *An Introduction to Reinforcement Learning*, MIT Press. 1998.

[9] P. Dayan, C. Watkins, "Q-learning," Machine Learning, 8, pp 279–292, 1992.

[10] Y. Engel, S. Mannor, R. Meir, "Reinforcement Learning with Gaussian Processes," In Proc. of the 22nd Int'l Conf. on Machine Learning. Vol. 22. Bonn, Germany, August 2005, pp. 201-208.

[11] M. Ghavamzadeh, & Y. Engel, "Bayesian policy gradient algorithms," Advances in Neural Information Processing Systems 19. Cambridge, MA: MIT Press, 2007.

[12] Jozef Hanc, Slavomir Tuleja, Martina Hancova, “Simple derivation of Newtonian mechanics from the principle of least action,” American Journal of Physics, Vol. 71. No. 4, April 2003, pp386 - 391.

[13] P. Deligne, D. S. Freed, “Classical field theory, Quantum fields and strings: a course for mathematicians,” Vol. 1, 2 (Princeton, NJ, 1996/1997), American Mathematical Society, Providence, RI, 1999, pp. 137–225.

[14] C.-E. Rasmussen and C.K.I Williams, *Gaussian Processes for Machine Learning*. MIT Press. 2006.

[15] S. Rajendran, M. Huber, “Learning to generalize and reuse skills using approximate partial policy homomorphisms,” International Conf. on Systems, Man and Cybernetics (SMC’09), pp 2239 - 2244, 2009.

[16] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman, “Efficient solution algorithms for factored MDPs,” Journal of AI Research, 19:399–468, 2003.

[17] C. Boutilier, R. Dearden, M. Goldszmidt, “Stochastic dynamic programming with factored representations,” Articial Intelligence, 121 (1-2), 49-107, 2000.

[18] B. Ravindran, and A. G. Barto, “Model minimization in hierarchical reinforcement learning,” In the Proc. of SARA, LNCS (2371), pp 196-211, 2002.

[19] M. Asadi and M. Huber, “Accelerating action dependent hierarchical Reinforcement Learning through autonomous subgoal discovery,” In ICML 2005, Bonn, Germany: ACM.

[20] P. Tadepalli and T. G. Dietterich, "Hierarchical explanation-based reinforcement learning. In Proc. 14th International Conference on Machine Learning," pp 358–366. Morgan Kaufmann, 1997.

[21] T. G. Dietterich, “An overview of MAXQ hierarchical reinforcement learning,” Lecture Notes in Artificial Intelligence. In Proceeding of 4th International Symposium, 2000, SARA 1864:26–44.

[22] B. Schölkopf and A. Smola. *Learning with Kernels. Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2002.

[23] B. Schölkopf, R. Herbrich, and A. J. Smola, “A generalized representer theorem,” Technical Report NC-TR-00-081, NeuroCOLT, 2000.

[24] U. von Luxburg, "A tutorial on spectral clustering," Technical Report 149, Max Planck Institute for Biological Cybernetics, 2006.

[25] A. Ng, M. Jordan, Y. Weiss, “On Spectral Clustering: Analysis and an Algorithm,” In Proc. of NIPS-14. MIT Press, 2001.

[26] F. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.

[27] S. X. Yu and J. Shi, “Multiclass spectral clustering,” In International Confererence on Computer Vision, 2003.

[28] M. Meila and J. Shi, “A random walks view of spectral segmentation,” In 8th International Workshop on Artificial Intelligence and Statistics (AISTATS), 2001.

[29] S. Mannor, I. Menache, A. Hoze, and U. Klein, “Dynamic abstraction in reinforcement learning via clustering,” In Proc. of the International Conference on Machine Learning, 2004, 21:560–567.

[30] C. V. Goldman and S. Zilberstein, “Decentralized control of control of cooperative systems: Categorization and complexity analysis,” JAIR, 22:143–174, 2004.

[31] D. Dolgov and E. Durfee, “Optimal resource allocation and policy formulation in loosely-coupled markov decision processes,” In AAMAS’04 pp 315–324, June 2004.

[32] I. I. Delice., and S. Ertugrul., “Intelligent modeling of human driver: a survey,” In Proc of the IEEE Intelligent Vehicle Symposium Istanbul, Turkey, pp 648-651, 2007.

[33] P. Dallaire, C. Besse, S. Ross, and B. Chaib-draa, “Bayessian reinforcement learning in continuous POMDPs with Gaussian processes,” In Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., pp 2604–2609, 2009.

[34] R. S. Sutton, D. Precup, and S. Singh, “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning,” Artificial Intelligence, 112:181–211, 1999.

[35] B. Ravindran and A. Barto, “SMDP homomorphisms: An algebraic approach to abstraction in Semi-Markov Decision Processes,” In Proc. of the 18th International Joint Conference on Artificial Intelligence, 2003.

[36] N. Aronszajn, “Theory of reproducing kernels,” Trans. American Mathematical Society, 686:337–404, 1950.

[37] P. Sollich, and C. K. I. Williams, "Using the equivalent kernel to understand gaussian process regression," In Advances in Neural Information Processing Systems 17, pp 1313–1320. MIT Press, Cambridge, MA, 2005.

[38] M. P. Wand, M. C., Jones. *Kernel Smoothing*. Chapman and Hall, London, 1994.

[39] J. O. Ramsay, B. W. Silverman. *Functional Data Analysis*. Springer New York NY, 2nd ed, 2005.

[40] F. Rossi, B. Conan-Guez, and A. El Golli, “Clustering functional data with the SOM algorithm,” In Proceedings of ESANN 2004, pages 305–312, Bruges, Belgium, April 2004.

[41] F. Rossi, N. Delannay, B. Conan-Guez, and M. Verleysen, “Representation of functional data in neural networks. Neurocomputing,” 64:183–210, March 2005.

[42] J. Rousu, C. Saunders, S. Szedmak, J. Shawe-Taylor, “Kernel based learning of hierarchical multilabel classification models,” *J. Machine Learning Research*, vol. 7, 2006.

[43] M. Medvedovic, S. Sivaganesan, "Bayesian infinite mixture model-based clustering of gene expression profiles," Bioinformatics, 18, pp 1194–1206, 2002.

[44] J. Q. Shi, B. Wang, "Curve prediction and clustering with mixtures of Gaussian process functional regression models," Statistics and Computing, 2008, 18(3): 267-283.

[45] Shi, J. Q. and B. Wang, "Curve prediction and clustering with mixtures of gaussian process functional regression models," Statistical Computing 18, 267–283, 2008.

[46] J. Shi and J. Malik, “Normalized cuts and image segmentation,” In IEEE Trans. on Pattern Analysis and Machine Intelligence, 22(8), 2000.

[47] M. Meila and J. Shi, “Learning segmentation with random walk,” In NIPS, 2001.

[48] P. K. Chan, M. D. F. Schlag, and J. Y. Zien, "Spectral k-way ratio-cut partitioning and clustering," IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems, 13(9):1088–96, 1994.

[49] H. Zha, C. Ding, M. Gu, X. He and H. Simon, “Spectral relaxation for K-means clustering,” Advances in Neural Information Processing Systems 14 (NIPS’01), 1057–1064, 2002.

[50] S. Boyd, L. Vandenberghe. *Convex Optimization*. Cambridge University Press, Cambridge 2004.

[51] D. P. Anderson. “BOINC: A System for Public-Resource Computing and Storage,” 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, PA, pp. 365-372, 2004.

[52] D Arthur, S Vassilvitskii, “Kmeans++: The Advantages of Careful Seeding,” ACM-SIAM Symposium on Discrete Algorithms, pp. 1027–1035, 2007.

[53] M. Avvenuti, P. Corsini, P. Masci, A. Vecchio, “Opportunistic Computing for Wireless Sensor Network,” IEEE Intl Conf. on Mobile Adhoc and Sensor Systems,” 2007, pp. 1-6.

[54] A. Bagherjeiran, C. F. Eick, C-S Chen, R. Vilalta, “Adaptive Clustering: Obtaining Better Clusters Using Feedback and Past Experience,” In Proc. Int. Conf. on Data Mining, 2005.

[55] Jim Basney, Miron Livny, and Todd Tannenbaum, “High Throughput Computing with Condor,” HPCU news, Volume 1(2), June 1997.

[56] M. Litzkow, M. Livny, and M. Mutka, “Condor - A Hunter of Idle Workstations,” In Proc. 8th Intl Conf. on Distributed Computing Systems, 1988, pp.104-111.

[57] Rajesh Raman, Miron Livny, and Marvin Solomon, “Matchmaking: Distributed Resource Management for High Throughput Computing,” Proc. of the 7th IEEE International. Symposium on High Performance Distributed Computing, July 28-31, 1998, Chicago, IL.

[58] D. Thain, T. Tannenbaum, and M. Livny, “Condor and the Grid. In Grid Computing: Making the Global Infrastructure a Reality,” John Wiley & ons Inc, 2002.

[59] T. T. Douglas Thain and M. Livny, “Distributed Computing in Practice: The Condor Experience,” Concurrency and Computation: Practice and Experience, 2004.

[60] J. Frey, T. Tannenbaum, M. Livny, “Condor-G: A Computation Management Agent for Multi-Institutional Grid”, Cluster Computing, Springer Netherlands, 2004, pp. 237-246.

[61] Condor manual, development release version 7.0. Available: http://www.cs.wisc.edu/condor/manual/

[62] I. Foster, Y. Zhao, I. Raicu, and S. Lu, “Cloud Computing and Grid Computing 360-Degree Compared,” Proc. IEEE Grid Computing Environments Workshop, pp. 1-10, 2008.

[63] I. Foster, C. Kesselman, “The Globus Project: A Status Report,” In Proc. Heterogeneous Computing Workshop, IEEE Press, 1998, pp. 4-18.

[64] I. Foster, C. Kesselman, S. Tuecke, “The Anatomy of the Grid: Enabling Scalable Virtual Organizations,” Intl. J. Supercomputer Applications, 2001.

[65] J. Chase, L. Grit, D. Irwin, J. Moore, and S. Sprenkle, “Dynamic Virtual Clusters in a Grid Site Manager,” In the 12th International Symposium on High Performance Distributed Computing (HPDC-12), 2003.

[66] M. Rodriguez, D. Tapiador, J. Fontan, E. Huedo, R.S. Montero, I.M. Llorente, “Dynamic Provisioning of Virtual Clusters for Grid Computing,” In Proceedings of the 3rd Workshop on Virtualization in High-Performance Cluster and Grid Computing (VHPC 08), in conjuction with EuroPar 08 (2008).

[67] Akihiko Konagaya, “The Grid as a ‘Ba’ for Biomedical Knowledge Creation,” Grid Computing in Life Science, LSGRID 2005, pp. 1-10.

[68] W. Emeneker and D. Stanzione, “Dynamic Virtual Clustering,” In Proc. Cluster, Austin, TX, September 2007.

[69] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, S. Tuecke, “A Resource Management Architecture for Metacomputing Systems,” Proc. IPPS/SPDP ’98 Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[70] Derrick Kondo, David P. Anderson and John McLeod VII, “Performance Evaluation of Scheduling Policies for Volunteer Computing,” 3rd IEEE International Conference on e-Science and Grid Computing. Bangalore, India, December 10-13, 2007.

[71] I. Sfiligoi et al., “The Pilot Way to Grid Resources Using glideinWMS,” In Proceedings of Computer Science and Information Engineering, 2009 WRI World Congress on, pp. 428-432, March 2009.

[72] Distributed.net: The First General-Purpose Distributed Computing Project. Available: http://www.distributed.net

[73] gLite, Lightweight Middleware for Grid Computing. Available: http://glite.web.cern.ch/glite/

[74] M. Papakhian, “Comparing Job-Management Systems: The User's Perspective,” IEEE Computational Science & Engineering, (April-June) 1998. Available: http://pbs.mrj.com

[75] S. Zhou, “LSF: Load Sharing in Large-Scale Heterogeneous Distributed Systems,” In Proceedings of the Workshop on Cluster Computing, 1992.

[76] Enabling Grids for E-science. Available: www.eu-egee.org

[77] CERN Twiki. http://twiki.cern.ch/twiki/bin/view/EGEE/BDII

[78] Open Science Grid. http://www.opensciencegrid.org

[79] Organization for the Advancement of Structured Information Standards, “Introduction to UDDI: Important Features and Functional Concepts,” 2004.

[80] W. T. Sullivan, III, D. Werthimer, S. Bowyer, J. Cobb, D. Gedye, D. Anderson, “A New Major SETI Project Based on Project SERENDIP Data and 100,000 Personal Computers. Astronomical and Biochemical Origins and the Search for Life in the Universe,” In Proc. of the Fifth Intl. Conf. on Bioastronomy. 1997.

[81] A. Tsaregorodtsev, V. Garonne, I. Stokes-Rees, "DIRAC: A Scalable Lightweight Architecture for High Throughput Computing," Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04), 2004, pp.19-25.

[82] K. Harrison, R.W.L. Jones, D.Liko, C.L. Tan, “Distributed Analysis in the ATLAS Experiment,” In Proc. AHM Conf., 2006.

[83] A. Klimentov, “ATLAS Distributed Data Management Operations. Experience and Projection,” Journal of Physics: Conf. Series, 2007.

[84] S. Kolos et al., “Online Monitoring software framework in the ATLAS experiment,” CHEP 2003, La Jolla, California, USA, 2003.

[85] T Maeno, “PanDA: Distributed Production and Distributed Analysis System for ATLAS,” Journal of Physics: Conference Series, 2008.

[86] P. Nilsson, “Experience from a Pilot based system for ATLAS,” Journal of Physics: Conference Series, 2008.

[87] P. Nilsson, J. Caballero, K. De, T. Maeno, M. Potekhin and T. Wenaus, “The PanDA system in the ATLAS experiment,” ACAT 2008 Conference Proceedings.

[88] B. DeWin, F. Piessens, W. Joosen, T. Verhanneman, “On the Importance of the Separation-Of-Concerns Principle in Secure Software Engineering,” In ACSA Workshop on the Application of Engineering Principles to System Security Design, 2003, pp. 1-10.

[89] D. Dolgov and E. Durfee, “Optimal resource allocation and policy formulation in loosely-coupled markov decision processes,” In AAMAS’04 pp 315–324, June 2004.

[90] A Galsyan, K Czajkowski, K Lerman, “Resource Allocation in the Grid Using Reinforcement Learning,” *Autonomous Agents and Multiagent Systems*, 2004.

[91] R. E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton.

[92] R. E. Bellman, “A Markovian Decision Process,” Journal of Mathematics and Mechanics, Vol 6, pp 679-684, 1957.

[93] K. Hsiao, L. Kaelbling, and T. Lozano-Pérez, “Grasping POMDPs,” in Proc. IEEE Int. Conf. on Robotics & Automation, 2007, pp. 4485–4692.

[94] P. Dallaire, C. Besse, S. Ross, and B. Chaib-draa, “Bayessian reinforcement learning in continuous POMDPs with Gaussian processes,” In Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst., pp 2604–2609, 2009.

[95] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller, “Context-specific independence in Bayesian networks,” In Proceedings of the Twelfth Conference on Uncertainty in Artificial Intelligence, pp 115–123, Portland, OR, 1996.

[96] C. Boutilier, R. Dearden, M. Goldszmidt, “Stochastic dynamic programming with factored representations,” Articial Intelligence, 121 (1-2), 49-107, 2000.

[97] C. Guestrin, D. Koller, R. Parr, and S. Venkataraman, “Efficient solution algorithms for factored MDPs,” Journal of AI Research, 19:399–468, 2003.

[98] Dabney, A. R. and Storey, J. D, “Optimality driven nearest centroid classificationfrom genomic data,” PLoS ONE, 2:e1002, 2007.

[99] I. I. Delice, and S. Ertugrul, “Intelligent modeling of human driver: a survey,” In Proc of the IEEE Intelligent Vehicle Symposium Istanbul, Turkey, pp 648-651, 2007.

[100] K Kozak, M Kozak, K Stapor, “Weighted K-Nearest-Neighbor Techniques for High Throughput Screening Data,” *Int. J. Biological, Biomedical and Medical Science* 1(3), 2006.

[101] N Lilith, K Dogancay, “Dynamic Channel Allocation for Mobile Cellular Traffic using Reduced-state Reinforcement Learning,” IEEE WCNC, vol. 4, pp. 2195-2200, 2004.

[102] R. H. Crites, A. G. Barto, “Elevator group control using multiple reinforcement learning agents,” Machine Learning 33:235-262, 1998.

[103] D. S. Bernstein, S. Zilberstein, “Reinforcement learning for weakly-coupled MDPs and an application to planetary rover control,” European Conference on Planning 2001.

[104] E. Kreyszig. *Introduction to Functional Analysis with Applications*. New York: Wiley, 1978.

[105] B. N. Singh, A. K. Tiwari, “Optimal selection of wavelet basis function applied to ECG signal denoising,” Digital Signal Processing, 16(3), pp 175-287, 2006.

[106] R. Coifman, M. Maggioni, “Diffusion wavelets,” Applied Computational Harmonic Analysis, 2005.

[107] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer: New York, 2007.

[108] C. Ding, X. He, “K-means clustering via principal component analysis,” In Proceedings of the International Machine Learning Conference (ICML). Morgan Kau man. 2004.

[109] I. Dhillon, Y. Guan, and B. Kulis, “Kernel k-means, spectral clustering and normalized cuts,” In Proc. 10th ACM KDD Conference, pp551–556, 2004.

[110] T. Xiangand S. Gong, “Spectral clustering with eigenvector selection,” Pattern Recognition, 41(3), 2008.

[111] F. Bach and M. Jordan, “Learning spectral clustering,” In Proc. of NIPS-16. MIT Press, 2004.

[112] I. Kondor and J. Lafferty, “Diffusion kernels on graphs and other discrete structures,” In Proceedings of ICML'02, 2002.

[113] A. Lazaric, M. Restelli, and A, Bonarini, “Reinforcement learning in continuous action spaces through sequential monte carlo methods,” Advances in Neural Information Processing Systems, 2007.

[114] S. Vijayakumar, A. D’Souza, T. Shibata, J. Conradt, and S. Schaal, "Statistical Learning for Humanoid Robots," Autonomous Robot, 12(1):55–69, 2002.

[115] D. Hardoon, J. Shawe-Taylor, “Decomposing the tensor kernel support vector machine for neuroscience data with structured labels,” Machine Learning, 79(1):1–18, 2009.

[116] H. Lodhi, J. Shawe-Taylor, N. Christianini and C. Watkins, “Text classification using string kernels,” NIPS (In Advances in Neural Information Processing Systems), Vol 13, 2001.

[117] J. Gruska. *Quantum Computing*. McGraw-Hill, New York, 1999.

[118] Satinder Singh, “Reinforcement Learning with a Hierarchy of Abstract Models,” In Proceedings of the National Conference on Artificial Intelligence, 1992.

[119] N. K. Jong, and P. Stone, “State Abstraction Discovery from Irrelevant State Variables,” In the Proc. of IJCAI, pp 752-757, 2005.

[120] G.D. Konidaris and A.G. Barto, "Skill discovery in continuous reinforcement learning domains using skill chaining," In Advances in Neural Information Processing Systems 22, pp 1015–1023, 2009.

BIOGRAPHICAL INFORMATION

Po-Hsiang Chiu, born in Taipei, Taiwan, planet Earth, receives a Bachelor of Science in Electrical Engineering from Tamkang Univesity in June 2000. He received a Masters in Computer Science from Syracuse University in December 2003, and a Ph.D. in Computer Science from University of Texas at Arlington in August 2011.

From September 2006 to January 2009, Po-Hsiang Chiu worked as a research assistant largely on site at the Brookhaven National Laboratory while proceeding with the doctorate program from UT at Arlington through distance education. Due to the special research opportunity at BNL in the field of Grid Computing technology driven by the ATLAS experiment, the author integrated his Grid-related research experience with his other research interest in Machine Learning thereby leading to the work in this disseration.

The author has been interested in the creation of the universe and metaphysical aspects of the complex, non-linear information field behind it. He believes that the research experience afforeded to him at UT at Arlington and BNL is just the beginning phase of a great advanture towards integrating various scientic disciplines for both techongical and spirtiual advacements. For reasons mentioned above, he will continue to pursue his research interests in Machine Learning and AI in general as well as his long-term passion in Astronomy, Astrophysics and perhaps the non-maintream science such as the study of extraterrestial life forms and our cosmic brothers, and quantum mechanical views of metaphysics, among others.